MASTER EN SCIENCES INFORMATIQUES

Structure pour la conception de spécifications d'objets

Liesenborghs, Jean-François; Mottet, Philippe

Award date:
1994

Awarding institution:
Universite de Namur

Facultés Universitaires Notre Dame de la Paix
Institut d'informatique
Rue Grandgagnage, 21, B - 5000 NAMUR

STRUCTURE POUR LA CONCEPTION DE SPECIFICATIONS D'OBJETS

Jean-François Liesenborghs
Philippe Mottet

Promoteur : *Pierre-Yves Schobbens*

Mémoire présenté en vue de l'obtention du
diplôme de Licencié et Maître en informatique

Année académique 1993-1994

Abstract

This thesis focuses on the analysis of the decomposition of object-oriented specifications using temporal logic. These decompositions are based on the Fiadeiro and Maibaum's theory. The thesis presents the horizontal and vertical decompositions : the first one allows to construct a complex system from simple objects in such way that the properties of the basic objects are preserved in the global object representing the system. The concurrency of the basic objects is managed within the global object. The vertical decomposition allows to translate an abstract object, thus not directly representable, in a more concrete object. This one have to be a faithful translation of the abstract object. The properties abided at the abstract level will still be at the concrete level.

Résumé

Dans ce mémoire, nous analysons des décompositions de spécifications orientées objets dans le cadre d'une logique temporelle. Ces décompositions sont basées sur la théorie de Fiadeiro et Maibaum. Nous allons présenter ce que l'on appelle la décomposition horizontale et la décomposition verticale : la première permet de construire à partir d'objets simples un système plus complexe tel que les propriétés des objets de base sont conservées dans l'objet global représentant le système. La concurrence des objets de base est gérée au sein de l'objet global. Le deuxième type de décomposition permet de traduire un objet abstrait, donc non implémentable directement, en un objet plus concret; ce dernier sera la traduction fidèle de l'objet abstrait. Les propriétés respectées au niveau abstrait le seront également au niveau concret.

Nous tenons à adresser nos vifs remerciements à notre promoteur Pierre-Yves Schobbens qui nous a guidé tout au long de l'élaboration de ce mémoire. Nous tenons également à remercier notre superviseur à l'étranger José Louis Fiadeiro, pour les explications et les conseils qu'il nous a donnés, ainsi que ses assistants pour le temps précieux qu'ils nous ont consacré. Nous adressons un remerciement tout particulier à Romina Demarco pour le temps passé à la correction de notre travail. Enfin nous remercions nos familles et nos amis pour le soutien moral qu'ils nous ont apporté.

Chapitre 1

Introduction

Dire que l'informatique, de nos jours, touche à tous les domaines est devenu un lieu commun. Pourtant cette discipline n'est qu'à ses balbutiements et beaucoup de progrès restent à faire pour qu'elle devienne une science à proprement parler. Un des fondements de l'informatique est la notion de spécification. Celle-ci a fait couler beaucoup d'encre car elle est la pierre d'achoppement de tout développement logiciel. Une bonne spécification permet d'éliminer les ambiguités et de définir de façon formelle les fonctionnalités du système élaboré.

Ces dernières années, une nouvelle approche de la spécification a fait son apparition : il s'agit de la spécification orientée objet. Cette nouvelle notion n'est pas un bouleversement des fondements de l'informatique mais elle permet de donner une structure plus claire et plus manipulable aux spécifications. Son impact n'est toutefois pas négligeable car cette forme d'approche a été très vite adoptée par une grande majorité du monde informatique. Cette notion dénote une tendance actuelle sous-jacente, qui est de vouloir réaliser des spécifications de haut niveau totalement indépendantes de la future implémentation et du système dans lequel elle sera réalisée.

Dans ce mémoire nous présenterons deux types de méthodes de décomposition d'objets, qui ont été développées par J.L. Fiadeiro et T. Maibaum. Ceux-ci se sont inspirés des travaux de Burtsall et Goguen qui ont, entre autres, créé le premier langage de spécification CLEAR et, des travaux de Pnueli qui initia l'utilisation de la logique temporelle pour spécifier les systèmes concurrents.

Les systèmes sont généralement spécifiés en terme d'entrées/sorties et rarement dans le cadre d'une logique temporelle. Dans ce document nous étudierons les systèmes réactifs; il est dès lors plus intéressant de réaliser les spécifications en logique temporelle, ce qui permet de décrire la vie d'un système et son évolution dans le temps suivant les événements qui se produisent.

Les deux types de décomposition présentées reprennent les trois concepts précités : langage de spécification, logique temporelle et spécification orientée objet.

La première méthode est appelée la décomposition horizontale. L'idée est de transformer le langage de spécification, qui consiste une suite parfois difficilement compréhensible de formules, en un ensemble plus structuré. Nous allons utiliser la spécification orientée objet pour décomposer la description, souvent complexe, d'un gros système, en descriptions de sous-systèmes qui seront plus abordables. Chaque sous-système sera un objet possédant un certain nombre d'actions qu'il pourra exécuter en réaction à un changement d'état ou suite à la demande d'un autre objet. Son état sera décrit par un groupe d'attributs (les variables d'un programme) qui ne pourront être modifiés que par les actions de l'objet. Ces objets constitueront les blocs qui, mis ensemble, formeront la spécification du système plus étendu.

Nous allons présenter une méthode pour assembler les spécifications de ces objets afin de former la spécification du gros système. Cette méthode permet de construire un nouvel objet plus complexe qui pourra lui-même être utilisé comme bloc d'une structure encore plus étendue. Cette technique est intéressante car elle permet une meilleure structuration mais elle est aussi intéressante du point de vue des démonstrations que l'on peut faire à propos d'un système. En effet, il est toujours plus facile de démontrer des propriétés vraies pour un petit objet plutôt que celles relatives à une grande structure. Les propriétés vraies pour les objets constituants, le seront toujours dans la super-structure, ce qui simplifie nettement les preuves à réaliser dans celle-ci.

La deuxième méthode sur laquelle nous allons nous attarder davantage est la décomposition verticale. Il s'agit d'utiliser certains principes de la première méthode (structure d'objets, composition d'objets, ...) pour décomposer une spécification abstraite en une spécification concrète. A l'aide de cette méthode, il sera donc possible de transformer une spécification de haut niveau en une spécification de bas niveau.

Nous allons clairement présenter ces deux approches en les illustrant à l'aide d'exemples que nous avons développés. Ceux-ci nous permettrons de dégager des conclusions générales sur ces décompositions et mettrons en évidence certaines améliorations qui peuvent être apportées. Pour chacune de ces améliorations, nous démontrerons que la cohérence de la théorie initiale est conservée. Nous avons également, dans ce travail, développé des systémes de démonstration pour prouver des propriétés à propos des objets. Ces modèles se basent sur des modèles existants que l'on a appliqués à notre type de spécification. Pour conclure, nous allons décrire une méthode générale de décomposition verticale, déduite à partir des exemples développés.

1.1 Pré-requis

Pour aborder ce document, une bonne connaissance de la logique et des spécifications algébriques est un atout essentiel. Certaines autres notions mathématiques telles que les morphismes peuvent faciliter la lecture. L'ensemble du document est illustré par des exemples simples permettant au lecteur de voir l'application pratique des concepts théoriques abordés.

1.2 Organisation du texte

Les trois premiers chapitres établissent les bases de notre travail. Le centre de notre analyse est contenu dans les chapitres 4, 5 et 6.

Le chapitre 2 présente les concepts théoriques de base que nous utiliserons. D'une part nous spécifions les concepts empruntés à la théorie des spécifications algébriques et d'autre part nous rappelons les grands principes de la logique temporelle en spécifiant les notations choisies.

Le chapitre 3 expose la théorie élaborée par J.L. Fiadeiro et T.Maibaum.

Le chapitre 4 développe un exemple d'implémentation basée sur la théorie exposée au Chapitre 3.

Le chapitre 5 détaille les modifications que nous avons apportées à la théorie initiale en démontrant leur correction et développe un exemple illustrant les modifications.

Le chapitre 6 donne une méthode générale s'appliquant aux différents cas pouvant être rencontrés dans les implémentations.

Chapitre 2

Concepts théoriques

2.1. Introduction

Dans ce chapitre nous présentons deux concepts théoriques qui sont à la base du développement des structures de spécifications décrites infra. Ces concepts théoriques sont d'une part, les spécifications algébriques et, d'autre part, la logique temporelle linéaire. Pour le premier concept, nous nous contentons de présenter les aspects qui nous seront utiles. En ce qui concerne la logique temporelle, nous donnons la syntaxe générale, en précisant celle que nous utiliserons, ainsi que la sémantique correspondant à cette syntaxe.

2.2. Les spécifications algébriques[1]

Les spécifications algébriques ont été utilisées pour spécifier des types de données connus et ont ensuite été étendues à la définition de types abstraits. Dans cette étude nous utiliserons certaines notions développées dans les spécifications algébriques. A l'aide de ces notions, nous spécifierons non pas des types de données mais des objets. Analysons dès lors deux concepts : la signature et l'algèbre.

---

[1]Les concepts sur les spécifications algébrique ont été tirés de [Ehrig & Marh 84] [Ehrig & Marh 90].

Prenons deux exemples de spécifications de types de données:

- la spécification de l'ensemble des booléens (BOOL);
- la spécification de l'ensemble des naturels(NAT).

BOOL

Sorts: BOOL

Symboles d'opération: True, false : $\rightarrow$ BOOL
Not : BOOL $\rightarrow$ BOOL
And : BOOL $\times$ BOOL $\rightarrow$ BOOL

Equations : b $\in$ BOOL
Not(True) = False
Not(Not(b)) = b
b and True = b
b and False = False

On obtient l'algèbre BOOL(B, true, false, $\neg$, $\wedge$) avec B={true, false}qui peut être enrichi par d'autres symboles d'opérations logiques tels que ($\vee$, $\rightarrow$, $\leftrightarrow$).

La spécification des nombres naturels avec le zéro, successeur et addition est donnée par:

NAT

Sorts : NAT

Symboles d'opération : 0 : $\rightarrow$ NAT
Add : NAT $\times$ NAT $\rightarrow$ NAT
Succ : NAT

Equations : n, m $\in$ NAT
Add(n,0) = n
Add(n,succ(m)) = succ(Add(n,m))

Dans ces spécifications, ce qui nous intéresse c'est la définition de l'ensemble des données et des opérations qui sont définies sur celles-ci. L'interprétation de cette définition nous sera aussi utile. Définissons donc la signature et son algèbre.

Définition 2.2.1.(Signature) :

$\Sigma = (S,\Omega)$ consiste en deux ensembles

- S ensemble des sorts
- Ω est l'ensemble des symboles d'opération avec comme arguments, les sorts $w \in S^*$ (concaténation de différents sorts) et, comme résultat, $s \in S$.

Définition 2.2.2.(Algèbre) :

$A = (S_A,\Omega_A)$ d'une signature $\Sigma = (S,\Omega)$ appelé aussi Σ-Algèbre est donnée par deux familles $S_A = (A_s)_{s\in S}$ et $\Omega_A = (N_A)_{N\in\Omega}$ où

- $\forall s \in S$, A_s sont des ensembles appelés ensembles de base au domaine de A.
- Pour chaque symbole de fonction $N \in \Omega_{s1,..,sn,s}$ avec $s1,..,sn \in S^+$ et $s \in S$, on a une fonction $N_A : A_{s1} \times A_{s2} \times .. \times A_{sn} \rightarrow A_s$ appelée opération sur A.

Remarque: nous n'avons pas défini les systèmes d'équations décrivant les opérations sur les types de données. Dans les spécifications d'objets telles que nous les envisagerons, ces équations sont supposées données par ailleurs, afin de ne pas alourdir les spécifications d'objets. De plus, les données manipulées par les objets seront des types de données classiques dont les opérateurs sont connus ou ont été définis dans de nombreux ouvrages.

2.3. La logique temporelle linéaire

2.3.1. Introduction[2]

"La logique temporelle linéaire permet de décrire des séquences de situations et de raisonner à leur sujet. On peut interpréter ces séquences de situations de plusieurs manières. Dans une interprétation proprement temporelle, la séquence de situations représente l'évolution de l'état du monde au cours du temps. Une autre interprétation que nous considérerons concerne le cas où la séquence de situations représente les états successifs d'un programme en cours d'exécution. Cette interprétation permet notamment d'utiliser la logique temporelle pour la spécification et la vérification de programmes.

[2]Cette introduction est tirée de [A.Thayse p179].

Il existe de nombreuses versions de la logique temporelle linéaire. Nous en présentons une qui a été très utilisée dans la spécification et la vérification de programmes parallèles. Ce domaine est probablement celui où la logique temporelle s'est avérée la plus utile en informatique."

2.3.2. Rappel de la notation BNF

La notation "Backus-Naur form" (BNF) est utilisée pour définir la syntaxe des formules que nous étudierons.
Cette notation peut être illustrée par la syntaxe de la logique standard des propositions, qui a comme catégorie principale, celle des *formules.*

<formule> ::= <formule atomique> | $\perp$ | <formule> $\rightarrow$ <formule>
Le symbole "::=" peut être lu comme "comprend" ou "consiste en" ou simplement "est". La barre verticale se lit "ou". L'équation veut donc dire qu'une formule est soit une formule atomique, soit faux($\perp$), ou une implication entre deux formules.

Si on a l'ensemble des formules atomiques noté Φ et une formule $p \in \Phi$, l'ensemble des formules générées à partir de Φ sera noté Fma(Φ).

La présentation de la syntaxe peut également être écrite

Formule atomique : $p \in \Phi$

Formule : $A, A_1, A_2 \in Fma(\Phi)$

$A ::= p \mid \perp \mid A_1 \rightarrow A_2$

2.3.3. Logique modale

La logique temporelle s'intègre dans le cadre de la logique modale. Nous allons avant toute chose définir les grands concepts de la logique modale.

2.3.3.1. Formules modales

Nous allons utiliser une logique du première ordre, cette logique est constituée au moyen de quatre sortes de symboles :

- les *variables* (notées x, y, z, etc.);
- les *constantes individuelles* (notées a, b, c, etc.);
- les *constantes fonctionnelles* ou *noms de fonctions* (notées f, g, h, <, >, etc.);
- les *constantes prédicatives* ou *noms de prédicat* (notées P, Q, etc.).

Ces symboles de base sont utilisés dans la formation des quatre concepts suivants.

• Un *terme* est une variable ou une forme fonctionnelle.

• Une *forme fonctionnelle* (opération) est la juxtaposition d'un symbole de constante fonctionnelle et d'un nombre adéquat de termes. Elle sera notée f (t_1, .., t_n) avec f un symbole de constante fonctionnelle, t_1, .., t_n des termes et $n \geq 0$. Si n=0, f( ) est un symbole de *constante individuelle*.

• Une *forme prédicative* est la juxtaposition d'un symbole de constante prédicative et d'un nombre adéquat de termes. Elle sera notée P (t_1, .., t_m) avec P un symbole de constante prédicative, t_1, .., t_m des termes et $m \geq 0$. Si m=0, P ( ) est une *proposition*.

• Un *atome* est une forme prédicative ou une égalité, c'est-à-dire une expression du type (s = t), où s et t sont des termes.

Comme nous avons des variables deux symboles supplémentaires sont utilisés $\forall$ et $\exists$. respectivement appelés *quantificateur universel* et *quantificateur existentiel.* Si x est une variable, $\forall x$ se lit "pour tout x" et $\exists x$ se lit "pour au moins un x"

Le langage de la logique modale des prédicats requiert un symbole supplémentaire, la "boîte" $\Box$.
Les lectures possibles de $\Box A$:

il est nécessairement vrai que A;
il est toujours vrai que A[3];
il faut que A;
on sait que A;
on croit que A;
après la terminaison du programme, A.

SYNTAXE

Si t est un terme
$t ::= x \mid f(t_1,..,t_n)$ où $n \geq 0$

Si A est une formule, :
$A ::= A \rightarrow A \mid \Box A \mid \bot \mid \forall x\, A \mid P(t_1,..,t_n)$

[3] Nous adopterons cette lecture.

Les différents connecteurs peuvent être définis à partir de ceux que nous avons :

Négation : $\neg A$ est $A \rightarrow \bot$
Vrai : $\top$ est $\neg\bot$
Disjonction : $A_1 \vee A_2$ est $(\neg A_1) \rightarrow A_2$
Conjonction : $A_1 \wedge A_2$ est $\neg(A_1 \rightarrow \neg A_2)$
Equivalence : $A_1 \leftrightarrow A_2$ est $(A_1 \rightarrow A_2) \wedge (A_2 \rightarrow A_1)$
"Diamant" : $\Diamond$ ou **F** [4] est $\neg\Box\neg A$
Il existe : $\exists x \,.\, A$ est $\neg\forall x \,.\neg A$

Schéma

Nous aurons souvent l'occasion de nous référer à un schéma, c'est-à-dire un ensemble de formules ayant toutes la même forme syntaxique.
Ex : $\Box A \rightarrow A$: ce schéma représente la collection des formules $\{\Box B \rightarrow B : B \in Fma(\Phi)\}$.

2.3.3.2. Frames et modèles

Un frame est une paire F=(S, R) où S est un ensemble non vide des mondes possibles, et R une relation binaire sur S :
de façon symbolique $R \subseteq S \times S$.
Un Φ-modèle sur un frame est un triplet M=(S, R, W, V) avec

- W un ensemble non vide d'individus
- V une fonction de valuation qui
  - associe à chaque symbole de constante fonctionnelle f à n arguments une fonction de W^n vers W.
  - associe à chaque couple constitué d'un élément s de S et d'une constante prédicative P à n arguments une fonction de W^n vers {vrai, faux}.

Prenons Φ comme l'ensemble des formules atomiques et Fma(Φ) l'ensemble des formules générées à partir de Φ.
La relation "A est vraie au point s dans le modèle M" notée

$$M \models_{ass,s} A$$

est définie inductivement sur la formation de $A \in Fma(\Phi)$ comme suit

---

[4] Dans ce mémoire le "diamant" sera toujours noté "**F**".

$M \models_{ass,s} p(t_1,..,t_n)$ ssi $V(p(t_1,..,t_n))(s) =$ vrai où $V(p(t_1,..,t_n))(s)$ peut être défini récursivement par

$$V(p(t_1,..,t_n)) = V(p)(v(t_1),..,v(t_n))$$

et $V(f(t_1,..,t_n)) = V(f)(v(t_1),..,v(t_n))$

et $V(x) = ass(x)$

$M \not\models_{ass,s} \perp$ car on n'aura jamais que $\perp$ est vrai à n'importe quel moment

$M \models_{ass,s} A1 \rightarrow A2$ ssi $M \models_{ass,s} A1$ implique $M \models_{ass,s} A2$

$M \models_{ass,s} \Box A$ ssi $\forall t \in S$, sRt implique $M \models_{ass,t} A$

$M \models_{ass,s} \forall x\, A$ ssi pour tout ass' $\sim_x$ ass, $M \models_{ass',s} A$

2.3.3.3. Vérité et validité

Une formule A est vraie dans un modèle ($M \models A$) si $M \models_{ass,s} A$ $\forall$ ass et $\forall s \in S$

A est valide dans un frame F=(S, R), noté $F \models A$ si

$M \models A$ $\forall$ modèle M=(S, R, V) basé sur F.

Encore plus généralement, on peut dire qu'un schéma ou un sous-ensemble de Fma est vrai ou valide (noté $M \models \Gamma$, et $F \models \Gamma$) où $\Gamma \subseteq$ Fma si tous les membres de Γ sont vrais dans M ou valides dans F.

2.3.4. La logique temporelle proprement dite

Après avoir défini la logique modale voyons ce que la logique temporelle apporte comme autres concepts.

2.3.4.1. Symboles

Nous avons vu les deux symboles "$\Box$" et "F" qui se lisent "désormais" et "tôt ou tard". Trois autres symboles sont souvent utilisés :

- O ou X[5] qui veut dire "à l'état suivant".
- $\mathcal{U}$ qui veut dire "jusqu'à ce que". C'est un symbole binaire tel que, pour $p, q \in \Phi$ on a
  $p \mathcal{U} q$ où p reste vrai jusqu'à ce que q soit vrai et q se produira tôt ou tard.
- $\mathcal{W}$ qui veut dire "à moins que". C'est un symbole binaire tel que pour $p, q \in \Phi$ on a
  $p \mathcal{W} q$ où p reste vrai, sauf si q arrive et q ne se produira pas forcément.

---

[5] Dans ce mémoire, nous utiliserons le symbole "**X**".

• **BEG**[6] qui dénote le début de la période étudiée et qui est une formule atomique temporelle telle que pour $p \in \Phi$: **BEG** $\rightarrow$ p veut dire que p sera vrai à l'instant 0.

Pour formaliser le sens de ces symboles, nous prenons une séquence d'états désignée par une paire F=(S,σ) où σ est une fonction N $\rightarrow$ S énumérant S comme une séquence $\sigma_0, \sigma_1,..,\sigma_n$
S sera l'ensemble des états.

Un modèle M=(S,σ,V) est défini de la manière habituelle.
$M \models_{ass,j} A$, veut dire "A est vraie dans le j^{è} état σ_j dans M" est défini par :

$M \models_{ass,j} p$ ssi $\sigma_j \in V(p(t1,..,tn))$
$M \not\models_{ass,j} \perp$
$M \models_{ass,j} A \rightarrow B$ ssi $M \models_{ass,j} A$ implique $M \models_{ass,j} B$
$M \models_{ass,j} OA$ ssi $M \models_{ass,j+1} A$
$M \models_{ass,j} \Box A$ ssi $\forall k \geq j \; M \models_{ass,k} A$
$M \models_{ass,j} A \mathcal{U} B$ ssi pour un $k \geq j \; M \models_{ass,k} B$ et
$\forall i$ tel que $j \leq i < k$, $M \models_{ass,i} A$
$M \models_{ass,j} A \mathcal{W} B$ ssi $\forall k \geq j \; M \models_{ass,k} A$ ou
$\exists i \geq j$ tel que $M \models_{ass,i} B$ et $j \leq m < i$ tel que $M \models_{ass,m} A$
$M \models_{ass,j}$ BEG ssi $j = 0$
$M \models_{ass,j} \forall x \; A$ ssi pour tout i et ass' $M \models_{ass',i} A$

2.3.7. Axiomatisation

Notre système axiomatique comporte :

(A0) toutes les instances de schémas valides de la logique propositionnelle.
(A1)$\mathbf{X}(A \rightarrow B) \rightarrow (\mathbf{X}A \rightarrow \mathbf{X}B)$
(A2)$\mathbf{X}\neg A \equiv \neg \mathbf{X}A$
(A3)$\mathbf{F}A \equiv \neg\Box\neg A$
(A4)$\Box(A \rightarrow B) \rightarrow (\Box A \rightarrow \Box B)$
(A5)$\Box A \rightarrow A \wedge \mathbf{X}\Box A$
(A6)$\Box(A \rightarrow \mathbf{X}A) \rightarrow (A \rightarrow \Box A)$
(A7)$A \mathcal{U} B \rightarrow \mathbf{F}B$
(A8)$A \mathcal{U} B \equiv B \vee (A \wedge \mathbf{X}(A \mathcal{U} B)$

[6]Ce symbole ne se trouve pas dans la logique temporelle classique, il est défini dans [Fiadeiro,Maibaum 92].

(A9)$A \mathcal{W} B \equiv B \vee (A \wedge \mathbf{X}(A \mathcal{W} B)$

(A10)$[C \wedge \Box(C \rightarrow (B \vee (A \wedge \mathbf{X}C)))] \rightarrow A \mathcal{W} B$

(A11)$\forall x \Box A(x) \equiv \Box \forall x A(x)$

(A12)$\exists x \Box A(x) \rightarrow \Box \exists x A(x)$

Chapitre 3

Présentation de la théorie initiale

3.1. Introduction

Ce chapitre introduit la théorie développée par J.L. Fiadeiro et T. Maibaum[7].Les deux premiers points nous permettront de voir comment spécifier un objet. Nous allons envisager les composants de manière individuelle et les relier afin d'obtenir un système complet (décomposition horizontale). Ensuite, nous verrons comment démontrer des propriétés concernant les systèmes. Enfin, nous verrons comment construire, à partir de ces notions, une méthode d'implémentation de spécifications abstraites d'objets en termes de spécifications concrètes (décomposition verticale).

3.2. Description de composants individuels

Ci-dessous nous présentons la méthode utilisée pour formaliser la spécification des objets. Chaque description d'objet est vue comme une théorie exprimée dans cette logique et est composée d'une signature et d'un ensemble de formules décrivant le comportement de l'objet. La signature de l'objet est essentielle car elle permet de définir le "langage" qui fait référence à l'objet.

---

[7] Elle est tirée des articles [Fiadeiro & Maibaum 92], [Fiadeiro & Maibaum 93A] et [Fiadeiro & Maibaum 93B].

3.2.1. Signature d'objet.

La signature[8] d'un objet doit comprendre au moins trois éléments : l'*univers*, les *structures d'attributs* et les *structures d'actions*. L'univers définit le contexte de données dans lequel l'objet est placé. Les attributs contiennent les informations qui décrivent l'état de l'objet. Les actions sont utilisées à la fois pour modifier des attributs et lors des interactions avec d'autres objets.

Définition 3.2.1.1. (Signature d'objet) :
Une *signature d'objet* est un triplet (Σ, A, Γ) où

- $\Sigma = (S,\Omega)$ est une signature[9] (l'*univers*) où S est un ensemble de sorts et Ω est une famille indexée[10] par $S^* \times S$ de symboles de fonctions;
- A est une famille indexée par $S^* \times S$ de symboles d'attributs. En termes de programmes, un symbole d'attribut d'arité nulle correspond à une variable;
- Γ est une famille indexée par S^* de symboles d'action. Les paramètres d'une action peuvent être les données nécessaires pour définir les effets des actions sur les attributs.

Les familles-indexées Ω, A, Γ portent sur des ensembles supposés disjoints et finis.
Nous allons définir deux ensembles $A^\bullet$, $\Gamma^\bullet$ qui nous seront utiles dans la théorie.
⇨ $A^\bullet = A(S^* \times S)$ est l'ensemble des symboles d'attribut de l'objet.
⇨ $\Gamma^\bullet = \Gamma(S^*)$ est l'ensemble des symboles d'action de l'objet.

Exemple 3.1.

Pour illustrer la description d'objets[11], nous envisageons l'exemple simplifié d'un gestionnaire de stock. Nous allons, dans un premier temps, décrire séparément le comportement de deux objets appelés "STOCK" et "COMMANDE". Ensuite, nous montrerons dans le point 3.3. comment mettre en relation ces deux composants, afin d'obtenir une description correcte de l'ensemble du système.

---

[8]Cette signature est une extension de la signature des spécifications algébriques présentée au chapitre 2.
[9]Définie dans [Ehrig & Marh 85].
[10]Une famille indexée est une fonction qui associe à chaque élément d'un ensemble, un ensemble d'éléments. Cette fonction est donc indexée par l'ensemble sur lequel elle porte.
[11]Approche proposée par J.L. Fiadeiro et T. Maibaum.

STOCK

Sorts : NAT, BOOL

Symboles d'opération : +, - : NAT × NAT → NAT
≤ : NAT × NAT → BOOL
0 : → NAT

Symboles d'attribut : Quantité : NAT
Nbresatisfaite : NAT

Symboles d'action : Approvisionner(NAT)
Enlever(NAT)

L'attribut "Quantité" représente l'état du stock. Les actions "Approvisionner" et "Enlever" permettent d'agir sur cet attribut. L'attribut "Nbresatisfaite" représente le nombre de commandes qui ont été exécutées et sera modifié lors de l'occurrence de l'action "Enlever".

COMMANDE

Sorts : NAT, QUEUE, BOOL

Symboles d'opération : filevide : QUEUE → BOOL
in : QUEUE × NAT → QUEUE
out : QUEUE → QUEUE
first : QUEUE → NAT
+, - : NAT × NAT → NAT

Symboles d'attribut : File : QUEUE
Qcommandée : NAT
Nbrereçue : NAT
Longueurfile : NAT

Symboles d'action : Ajouter(NAT)
Retirer(NAT)

"File" et "Qcommandée" sont les attributs de l'objet "COMMANDE", ils représentent respectivement la liste des commandes passées et non encore exécutées ainsi que la quantité de la commande en cours.

Cet objet possède également les attributs "Nbrereçue" et "Longueurfile" qui sont, d'une part, le nombre de commandes déjà reçues et, d'autre part, la longueur de la file des commandes non encore exécutées. Les deux actions de cet objet sont "Ajouter" et "Retirer". La façon dont elles agissent sur les attributs est détaillée dans la section suivante.

Une structure d'interprétation pour un objet est donnée par un algèbre qui interprète les paramètres de l'univers, une projection qui donne les valeurs prises par les attributs à chaque instant et, une projection qui donne les actions qui s'exécutent à chaque instant.

Définition 3.2.1.2. (θ-structures d'interprétation)
Une θ-structure d'interprétation I pour une signature $\theta = (\Sigma, A, \Gamma)$ est un triplet $(\mathrm{U},\mathrm{A},\mathrm{G})$ où:

- U est une Σ-algèbre, c'est à dire qu'elle assigne à tout symbole $s \in S$ un ensemble s_U, et à chaque symbole de fonction $f \in \Omega_{<s_1,...,s_n>,s}$, une fonction $f_U : s_{1U} \times ... \times s_{nU} \rightarrow s_U$;
- A projette $f \in A_{<s_1,...,s_n>,s}$ vers $\mathrm{A}(f) : s_{1U} \times ... \times s_{nU} \times N_0 \rightarrow s_U$. Nous utiliserons la notation $\mathrm{A}(f)(i)$ pour $i \in N_0$ afin de dénoter la fonction $A(f)(b_1,...,b_n,i)$.
- G projette $g \in \Gamma_{<s_1,...,s_n>}$ vers $\mathrm{G}(g) : s_{1U} \times ... \times s_{nU} \rightarrow P(N_0)$.

Où N_0 représente l'ensemble des naturels sans le zéro. Nous prenons les nombres naturels pour représenter le domaine temporel, c'est à dire que nous considérons un temps linéaire et discret. La projection A donne la valeur des attributs à chaque instant. On dénote une action par l'ensemble des instants durant lesquels l'action s'exécute.

3.2.2. Description des objets

Pour décrire le comportement des objets, nous utilisons cinq formules typiques :

a) **BEG** $\rightarrow$ **X**condition[12].

Ce type d'axiomes exprime les conditions qui doivent être vérifiées à la naissance de l'objet. Ce genre d'axiomes définit la propriété de Début des objets.

[12]Les conditions dans les formules sont des formules du premier ordre sur les attributs de l'objet.

b) action $\wedge$ condition $\rightarrow$ **X**condition.

Ce type d'axiomes spécifie les effets qu'a une action sur les attributs. Ce genre d'axiomes définit la propriété de Changement des objets ***(Change property)***.

c) action $\rightarrow$ condition.

Ce type d'axiomes spécifie les restrictions que l'on veut mettre sur l'occurrence des actions. Ils sont à la base de la propriété de Sécurité des objets (***Safety property***).

d) condition $\rightarrow$ **F**action.

Ce type d'axiomes impose que, si une certaine condition est respectée, une action sera exécutée tôt ou tard. Ils sont à la base de la propriété de Vivacité des objets ***(Liveness property)***.

e) action $\wedge$ condition $\rightarrow$ action

Ces axiomes[13] permettent de synchroniser les actions entre elles. Ce sont les axiomes de Synchronisation de l'objet.

Il existe aussi sur les objets une propriété générale très importante, la propriété de localité. Celle-ci se révèle très utile lors des démonstrations car elle garantit le fait que les attributs d'un objet ne peuvent être modifiés que par l'occurrence des actions de cet objet.
Pour toute signature $\theta = (\Sigma, A, \Gamma)$ nous avons la formule :

$$\left(\bigvee_{g \in \Gamma^{\bullet}} (\exists x_g) g(x_g) \right) \vee \left(\bigwedge_{a \in A^{\bullet}} (\forall x_a)(Xa(x_a) = a(x_a)) \right)$$

où pour chaque symbole u, x_u est un vecteur de variables, toutes distinctes, des sorts appropriés.

[13]Ces axiomes ne font pas partie de la théorie initiale. C'est en développant l'exemple, présenté au chapitre 4, que nous avons constaté leur nécessité.

Cette formule traduit le fait que soit, une action se produit, soit aucune valeur d'attribut ne change. Cette propriété fait partie intégrante de la logique, elle n'est pas reprise dans la description des objets.

Cet ensemble de formules (localité exceptée), ajouté à la signature de l'objet, nous donne la description complète d'un objet. Cela peut être formalisé par la définition suivante :

Définition 3.2.2.1 (description) : une description d'objet est une paire (θ, Φ), où θ est la signature d'un objet et Φ est un ensemble (fini) de θ-formules (les axiomes de la description).

Exemple 3.1.
La description complète des objets "STOCK" et "COMMANDE" devient donc:

STOCK

Sorts : NAT, BOOL

Symboles d'opération : +, - : NAT × NAT → NAT
≤ : NAT × NAT → BOOL
0 : → NAT

Symboles d'attribut : Quantité : NAT
Nbresatisfaite : NAT

Symboles d'action : Approvisionner(NAT)
Enlever(NAT)

Axiomes :

{axiomes de Début de l'objet}

S1) **BEG** → Quantité = 0
S2) **BEG** → Nbresatisfaite = 0

{ axiomes de Changement de l'objet}

S3) Approvisionner(x) → **X**Quantité = Quantité + x
S4) Enlever(x) → **X**Quantité = Quantité - x
S5) Enlever(x) → **X**Nbresatisfaite = Nbresatisfaite + 1
S6) Approvisionner(x) → **X**Nbresatisfaite = Nbresatisfaite

{ axiomes de Sécurité de l'objet}

S7) Enlever(x) → (x ≤ Quantité) = true

{ axiomes de vivacité de l'objet}

S8[14]) (Quantité ≤ 10)=true → ∃x **F**Approvisionner(x)

COMMANDE

Sorts : NAT, QUEUE, BOOL

Symboles d'opération : filevide : QUEUE → BOOL
in : QUEUE × NAT → QUEUE
out : QUEUE → QUEUE
first : QUEUE → NAT
+, - : NAT × NAT → NAT

Symboles d'attribut : File : QUEUE
Qcommandée : NAT
Nbrereçue : NAT
Longueurfile : NAT

Symboles d'action : Ajouter(NAT)
Retirer(NAT)

Axiomes : C1) **BEG** → filevide(File) = true
C2) **BEG** → Longueurfile = 0
C3) **BEG** → Nbrereçue = 0

C4) Ajouter(x) → **X**File = in(File,x)
C5) Ajouter(x) → **X**Longueurfile = Longueurfile + 1
C6) Ajouter(x) → **X**Nbrereçue = Nbrereçue + 1
C7) Ajouter(x) → **X**Qcommandée = Qcommandée
C8) Retirer(x) → **X**File = out(File)
C9) Retirer(x) → **X**Qcommandée = x
C10) Retirer(x) → **X**Longueurfile = Longueurfile - 1
C11) Retirer(X) → **X**Nbrereçue = Nbrereçue

[14]Dans l'axiome S8, nous avons imposé que dès que le stock devient inférieur à dix unités, il faut réapprovisionner. Cet axiome peut être modifié afin de fixer un seuil différent.

C12) Retirer(x) → x = first(File)
C13) Retirer(x) → filevide(File) = false

C14) filevide(File) = true → ∃x **F**ajouter(x)
C15) filevide(File) = false → ∃x **F**Retirer(x)

Les éléments qui composent le système étant définis, il nous reste à spécifier son fonctionnement.

3.3 Description d'un système

Nous venons de voir comment spécifier des objets sous forme de théorie. Il reste maintenant à voir comment réunir ces objets.

3.3.1 La notion de morphisme

Pour formaliser les liens entre les objets nous faisons appel à la notion de morphisme. Présentons d'abord cette notion de façon théorique et ensuite nous verrons comment l'utiliser pour faire le lien entre les objets.

Les morphismes sont considérés dans notre approche comme des relations entre des descriptions d'objets. Leur définition se présente de la manière suivante :

Définition 3.3.1.1. (Morphisme de signature) :
Etant donné deux signatures d'objets $\theta_1 = (\Sigma_1, A_1, \Gamma_1)$ et $\theta_2 = (\Sigma_2, A_2, \Gamma_2)$, un morphisme σ de θ_1 vers θ_2 consiste en :

- un morphisme de signatures algébriques $\sigma_v : \Sigma_1 \rightarrow \Sigma_2$;
- pour chaque $f : s_1,...,s_n \rightarrow s$ dans A_1 un symbole d'attribut $\sigma_\alpha(f) : \sigma_v(s_1),..s_1\sigma_v(s_n) \rightarrow \sigma_v(s)$ dans A_2;
- pour chaque $g : s_1,...,s_n$ dans Γ_1 un symbole d'action $\sigma_\gamma(f) : \sigma_v(s_1),..,\sigma_v(s_n)$ dans Γ_2 .

Un morphisme entre deux descriptions (θ_1, Φ_1) et (θ_2, Φ_2) identifie les symboles dans θ_2 qui sont utilisés pour interpréter les symboles de θ_1 et implique que pour chaque formule p, pour laquelle $(\Phi_1 \Rightarrow_{\theta 1} p)$ est valide, $(\Phi_2 \Rightarrow_{\theta 2} \sigma(p))$ est valide. De même, pour un ensemble arbitraire de θ_1-formule Φ et une θ_1-formule p, si $(\Phi \Rightarrow_{\theta 1} p)$ est valide, alors

$(\sigma(\Phi) \Rightarrow_{\theta 2} \sigma(p))$ doit aussi être valide.

Comme nous l'avons vu précédemment, l'axiome de localité n'est pas mis explicitement dans la description des objets. Or, toutes les formules valides pour θ_1 doivent avoir une image valide pour θ_2, nous allons étendre la notion de morphisme de telle manière que les conditions de localité des différents composants soient préservées dans leur nouvel environnement.

$$\Big(\bigvee_{g\in\Gamma^\bullet}(\exists x_g)g(x_g)\Big)\vee\Big(\bigwedge_{a\in A^\bullet}(\forall x_a)(Xa(x_a)=a(x_a))\Big)$$

qui est un théorème de (θ_1, Φ_1) devra être repris sous la forme

$$\sigma\Big(\Big(\bigvee_{g\in\Gamma^\bullet}(\exists x_g)g(x_g)\Big)\vee\Big(\bigwedge_{a\in A^\bullet}(\forall x_a)(Xa(x_a)=a(x_a))\Big)\Big)$$

dans la description de (θ_2, Φ_2), ce qui sera noté en abrégé $\theta_1 \xrightarrow{\sigma} \theta_2$.

Cette traduction de la formule de localité des objets va se révéler très utile lors des démonstrations.

Définition 3.3.1.2 (Morphisme) : Etant donné deux descriptions d'objets (θ_1, Φ_1) et (θ_2, Φ_2), un morphisme $\sigma : (\theta_1, \Phi_1) \rightarrow (\theta_2, \Phi_2)$ est un morphisme de signatures $\sigma : \theta_1 \rightarrow \theta_2$ tel que :

- $(\Phi_2 \Rightarrow_{\theta 2} \sigma(p))$ est valide pour tout $p \in \Phi_1$
- $(\Phi_2 \Rightarrow_{\theta 2} \theta_1 \xrightarrow{\sigma} \theta_2)$ est valide

3.3.2 Création de liens entre descriptions d'objets (Canaux)

Le lien entre deux objets peut se faire en créant des relations entre ceux-ci. Ces relations vont porter sur les actions des objets. L'idée est de créer un objet qui permettra de représenter le lien entre deux ou plusieurs autres. Le lien se fera par le biais de morphismes déterminant les relations entre les actions de ce nouvel objet et les actions à synchroniser dans les objets à relier.

Voyons, à l'aide de notre exemple, comment cela prend forme en pratique. Dans notre système les deux objets "STOCK" et "COMMANDE" vont interagir par le biais de leurs actions respectives "Enlever" et "Retirer". Nous synchronisons ces deux actions en construisant un nouvel objet nommé "CANAL", qui possède une action unique "Lier". Relions ce nouvel objet aux deux objets de départ par deux morphismes "Reçoit" et "Envoie". L'un de ces morphismes établit la relation entre l'action "Lier" de "CANAL" et "Enlever" de "STOCK"; l'autre établit la relation entre l'action "Lier" et l'action "Retirer" de "COMMANDE".

Schématiquement, cela nous donnera :

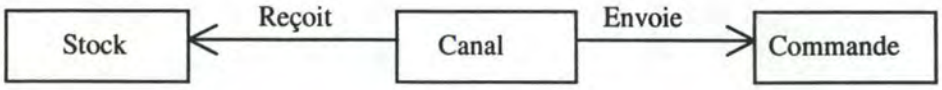

Fig.3.1. : *Schéma du lien entre "STOCK" et "COMMANDE".*

La description formelle de ce nouvel objet et des deux morphismes est la suivante:

Canal

Sorts : NAT

Symboles d'opération : /

Symboles d'attribut : /

Symboles d'action : Lier (NAT)

Axiomes : /

Reçoit

Relation: Canal → Stock
Sorts : ID_{NAT}, ID_{BOOL}
Symboles d'action : Lier(x) →Enlever(x)

Envoie

Relation: Canal → Commande
Sorts : ID_{NAT}, ID_{BOOL}
Symboles d'action : Lier(x) → Retirer(x)

Le lien étant créé entre les deux objets, nous allons voir comment obtenir un objet dont la description nous donnera le fonctionnement du système global.

3.3.3 Sommer deux objets (Colimite[15])

Nous ne devons pas perdre de vue que notre but est de formaliser un système, c'est pourquoi il nous faut obtenir un objet qui décrit le comportement du système. Cet objet sera le résultat de l'amalgame des objets composants. Faisont la "somme"[16] de la description des objets constituant le système pour obtenir sa spécification. Un objet "résultat" est créé, celui-ci possède comme attributs tous les attributs des objets sommés. Il existe donc une relation bijective entre l'union des attributs des objets sommés et l'ensemble des attributs de l'objet résultat. Pour les actions, il faut considérer deux familles distinctes: les actions non synchronisées et les actions synchronisées. Chaque action non synchronisée aura une image dans l'objet résultat et il existera une relation bijective entre cet ensemble d'images et l'union des actions non synchronisées des objets sommés. Les actions synchronisées auront elles le même symbole d'action dans l'objet "résultat".

Eclairons nos propos en reprenant notre exemple. A partir de la structure que nous avons obtenue dans la section précédente, nous réalisons la "somme" des deux objets mis en relation.
Nous créons l'objet "SYSTEME" qui aura trois symboles d'action, "Livrer", qui est l'image à la fois de "Retirer" et de "Enlever", "Enregistrer" qui est l'image de "Ajouter" et "Stocker" qui est l'image de "Approvisionner".
Ce qui donne sous forme schématique :

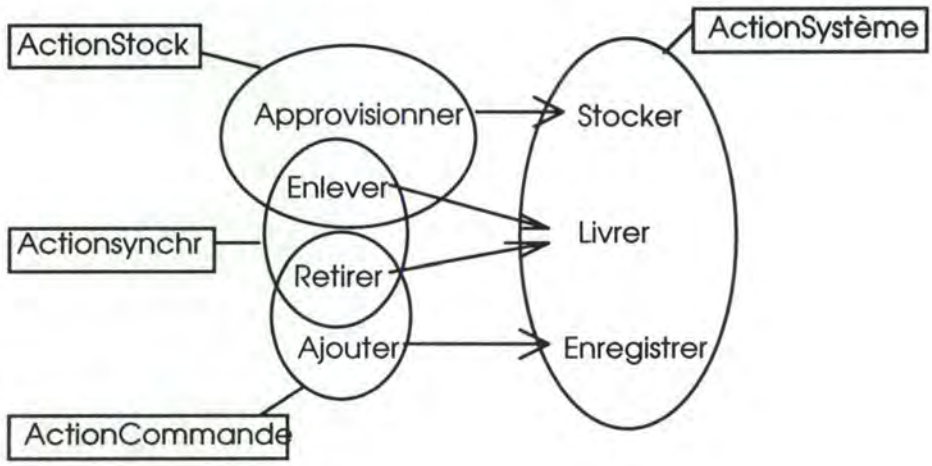

Fig.3.2. : *Relations entre les actions de "STOCK" et "COMMANDE" et les actions de "SYSTEME".*

---

[15] Terme emprunté à la théorie des catégories, sur laquelle se base notre "somme" d'objet.
[16] Cette "somme" a été définie dans [Fiadeiro & Maibaum 92].

Les attributs de "SYSTEME" seront "Réserve", "Cderéalisée", "Liste", "Qcdée", "Nbrcde", "Longueurliste" qui sont les images de chacun des attributs des deux objets de départ. Schématiquement cela donne :

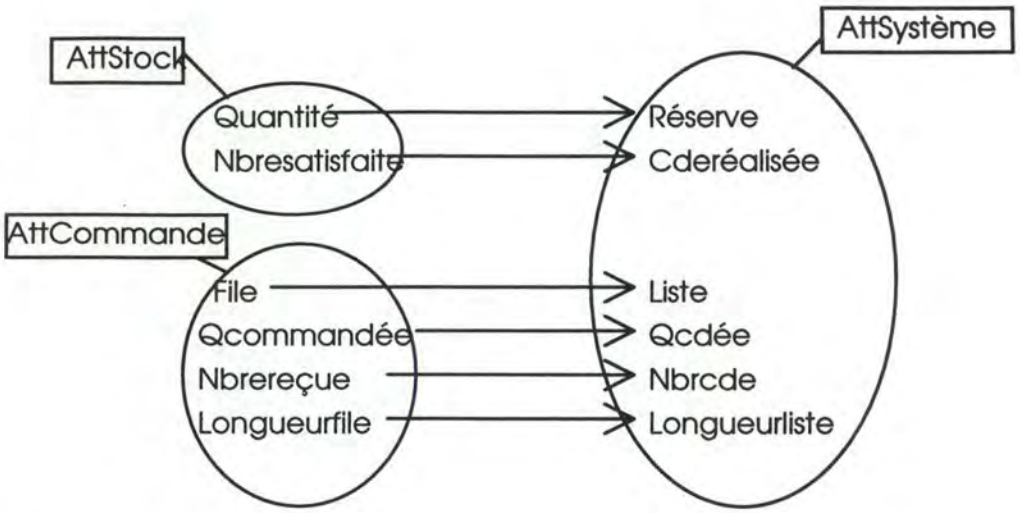

Fig.3.3. : *Relations entre les attributs de "STOCK" et "COMMANDE" et les attributs de "SYSTEME".*

L'objet "SYSTEME" est la somme (la colimite) des deux objets "STOCK" et "COMMANDE".

SYSTEME

Sorts : NAT, QUEUE, BOOL

Symboles d'opération : filevide : QUEUE → BOOL
in : QUEUE × NAT → QUEUE
out : QUEUE → QUEUE
first : QUEUE → NAT
+, - : NAT × NAT → NAT
≤, ≥ : NAT × NAT → BOOL
0 : → NAT

Symboles d'attribut : Réserve : NAT
Cderéalisée : NAT
Liste : QUEUE
Qcdée : NAT
Nbrcde : NAT
Longueurliste : NAT

Symboles d'action : Livrer (NAT)
Enregistrer (NAT)
Stocker (NAT)

Axiomes :

{Axiomes de "COMMANDE"}

ST1) **BEG** → filevide(Liste) = true
ST2) **BEG** → Longueurliste = 0
ST3) **BEG** → Nbrcde = 0

ST4) Enregistrer(x) → **X**Liste = in(Liste,x)
ST5) Enregistrer(x) → **X**Longueurliste = Longueurliste + 1
ST6) Enregistrer(x) → **X**Nbrcde = Nbrcde + 1
ST7) Enregistrer(x) → **X**Qcdée = Qcdée
ST8) Livrer(x) → **X**Liste = out(Liste)
ST9) Livrer(x) → **X**Qcdée = x
ST10) Livrer(x) → **X**Longueurliste = Longueurliste - 1
ST11) Livrer(X) → **X**Nbrcde = Nbrcde

ST12) Livrer(x) → x = first(Liste)
ST13) Livrer(x) → filevide(Liste) = false

ST14) filevide(Liste) = true → ∃ x **F**Enregistrer(x)
ST15) filevide(Liste) = false → ∃ x **F** Livrer(x)

ST16) ¬∃x (Enregistrer(x) ∨ Livrer(x)) → **X**Liste = Liste ∧ **X**Qcdée = Qcdée ∧ **X**Nbrcde = Nbrcde ∧ **X**Longueurliste = Longueurliste

{Axiomes de "STOCK"}

ST17) **BEG** → Réserve = 0
ST18) **BEG** → Cderéalisée = 0

ST19) Stocker(x) → **X**Réserve = Réserve + x
ST20) Livrer(x) → **X**Réserve = Réserve - x
ST21) Livrer(x) → **X**Cderéalisée = Cderéalisée + 1
ST22) Stocker(x) → **X**Cderéalisée = Cderéalisée

ST23) Livrer(x) → (x ≤ Réserve) = true

ST24) Réserve = 0 → ∃x **F**Stocker(x)

ST25) ¬∃x (Stocker(x) ∨ Livrer(x)) → **X**Reserve = Réserve ∧ **X**Cderéalisée = Cderéalisée

Remarque : dans notre exemple, les noms des actions et des attributs de l'objet "SYSTEME" sont différents des noms des attributs et des actions des objets "COMMANDE" et "STOCK". Ce changement de nom n'est pas obligatoire mais sert à montrer que les symboles ne sont que des représentations et que l'essentiel, c'est la correspondance entre les attributs et les actions représentées par ces symboles.

La définition de la somme telle que nous l'avons donnée sous-entend qu'il existe entre chaque objet sommé et l'objet résultat un morphisme.

Dans notre exemple nous obtenons deux morphismes supplémentaires que nous nommerons "Gestion" et "St", le schéma global est donc :

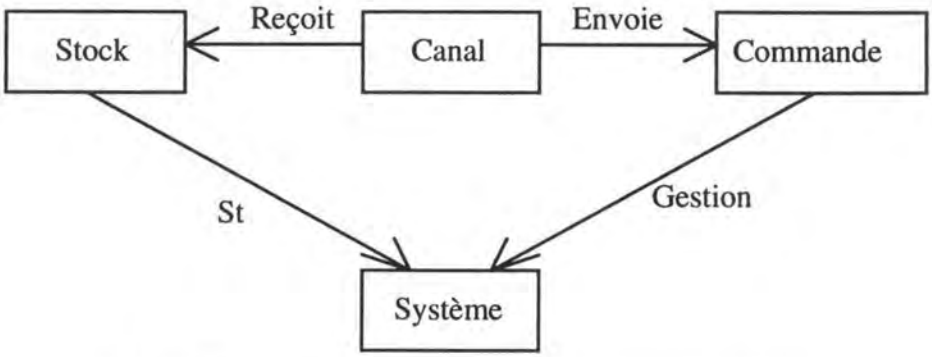

Fig. 3.4. : *Schéma final montrant les différents liens entre les objets.*

Gestion

Relation : Commande → Système
Sorts : ID_{NAT}, ID_{BOOL}
Symboles d'action : Retirer(x) → Livrer(x)
Symboles d'attribut : File → Liste
Qcommandée → Qcdée
Nbrereçue → nbrcde
Longueurfile → Longueurliste

St

Relation : Stock → Système
Sorts : ID_{NAT}, ID_{BOOL}, ID_{QUEUE}
Symboles d'action : Enlever(x) → Livrer(x)

Symboles d'attribut :	Quantité → Réserve Nbresatisfaite → Cderéalisée

3.4 Démonstration de propriétés

Il serait intéressant d'établir une méthode qui nous permette de prouver certaines propriétés des objets. Une fois cette méthode établie nous pourrons déterminer un invariant pour l'objet "SYSTEME" et démontrer qu'il est toujours respecté.

3.4.1 Système de démonstration

Pour démontrer qu'une propriété "P" est vraie dans un objet, nous prouvons deux choses:

1. "P" est vraie à la naissance de l'objet ce qui revient à démontrer :

   BEG → P

2. "P" est vraie durant la vie de l'objet autrement dit si "P" est vraie à un instant donné, alors, quel que soit l'événement, "P" est vraie à l'instant suivant. Ce qui revient à démontrer la formule :

   ∀ e ∈ événement : e → (P → **X**P)

Prenons une propriété simple qui semble intuitivement vraie pour un des objets de notre exemple. L'objet "STOCK" semble respecté la propriété *(Quantité ≥ 0) = true*. Appliquons le raisonnement pour démontrer de façon formelle cette propriété.

Il nous faut démontrer :

a) STOCK ⇒ (**BEG** → (Quantité ≥ 0) = true)
b) STOCK ⇒ ((Quantité ≥ 0) = true → **X**((Quantité ≥ 0) = true))

Le point a) se démontre de façon trivial par l'axiome S1 de "STOCK".

Pour démontrer b), il faut, comme nous l'avons vu, démontrer que, quel que soit l'événement, la propriété reste vraie. Dans "STOCK", les événements possibles sont la survenance d'une de ses actions "Enlever" ou "Approvisionner", ou l'absence d'exécution d'action (il ne se passe rien). Trois formules sont donc à démontrer :

b.1) Enlever(x) → ((Quantité ≥ 0) = true → **X**((Quantité ≥ 0) = true))
b.2) Approvisionner(x) → ((Quantité ≥ 0) = true → **X**((Quantité ≥ 0) = true))
b.3) ¬∃x (Approvisionner(x) ∨ Enlever(x)) → ((Quantité ≥ 0) = true → **X**((Quantité ≥ 0)) = true)

De la définition habituelle des opérations "+" et "-" pour les naturels, nous pouvons déduire deux axiomes.

$(x \leq q) = true \rightarrow ((q - x) \geq 0) = true$

$((x \geq 0) = true) \wedge ((q \geq 0) = true) \rightarrow ((x + q) \geq 0) = true$

Ceux-ci seront utilisés dans la preuve.

La démonstration de b.1) est :

1. Enlever(x) → (x ≤ Quantité) = true	S1
2. Enlever(x) → **X**Quantité = Quantité - x	S2
3. (x ≤ q) = true → ((q - x) ≥ 0) = true	déf. de "-" pour NAT
4. Enlever(x) → ((Quantité ≥ 0) = true → X((Quantité ≥ 0) = true))	**1., 2. et 3.**

La démonstration de b.2) est:

1. Approvisionner(x) → **X**Quantité = Quantité + x	S3
2. ((x ≥ 0) = true) ∧ ((q ≥ 0) = true) → ((x + Q) ≥ 0) = true	déf. de "+" pour NAT
3. Approvisionner(x) → ((Quantité ≥ 0) = true → X((Quantité ≥ 0) = true))	**1. et 2.**

b.3) se démontre par l'axiome de localité.

3.4.2 Démonstration d'un invariant

Dans notre exemple un invariant de l'objet "SYSTEME" est :

Cderéalisée = Nbrcde - Longueurliste

D'après notre système de démonstration nous avons à démontrer :

a) SYSTEME ⇒ (**BEG** → Cderéalisée = Nbrcde - Longueurliste)
b) SYSTEME ⇒ (Cderéalisée = Nbrcde - Longueurliste →
X (Cderéalisée = Nbrcde - Longueurliste))

Ici aussi le a) se démontre de façon triviale par les axiomes ST2, ST3 et ST18.

Pour démontrer b) il faut isoler tous les cas possibles, ce qui nous donne :

b.1) Livrer(x) → (Cderéalisée = Nbrcde - Longueurliste → **X** (Cderéalisée = Nbrcde - Longueurliste))

b.2) Enregistrer(x) → (Cderéalisée = Nbrcde - Longueurliste →
X (Cderéalisée = Nbrcde - Longueurliste))

b.3) Stocker(x) → (Cderéalisée = Nbrcde - Longueurliste → **X** (Cderéalisée = Nbrcde - Longueurliste))

b.4) ¬∃x (Livrer(x) ∨ Enregistrer(x) ∨ Stocker(x)) → (Cderéalisée = Nbrcde - Longueurliste → **X** (Cderéalisée = Nbrcde - Longueurliste))

La démonstration de b.1) est :

1. Livrer(x) → **X**Longueurliste = Longueurliste - 1	ST10
2. Livrer(x) → **X**Nbrcde = Nbrcde	ST11
3. Livrer(x) → **X**Cderéalisée = Cderéalisée + 1	ST21
4. Livrer(x) → (Cderéalisée = Nbrcde - Longueurliste → X (Cderéalisée = Nbrcde - Longueurliste))	**1., 2. et 3.**

Pour démontrer b.2), il faut tenir compte du fait que nous nous trouvons dans un système concurrent, plusieurs actions peuvent donc se dérouler simultanément.

Quatre cas sont a considérer :

- les actions "Enregistrer", "Livrer" et "Stocker" se déroulent en même temps;
- les actions "Enregistrer" et "Livrer" se déroulent en même temps;
- les actions "Enregistrer" et "Stocker" se déroulent en même temps ou seule l'action "Enregistrer" se déroule .

Nous avons déjà démontré que si "Livrer" s'exécute et si nous avons l'invariant, nous l'aurons toujours à l'instant suivant. Les deux premiers cas sont donc prouvés.

Voyons ce qui se passe dans les deux derniers cas.

1. Enregistrer(x) → **X**Longueurliste = Longueurliste + 1	ST5
2. Stocker(y) → **X**Cderéalisée = Cderéalisée	ST22
3. Enregistrer(x) → **X**Nbrcde = Nbrcde + 1	ST6
4. Enregistrer(x) ∧ Stocker(y) → (Cderéalisée = Nbrcde - Longueurliste → **X**(Cderéalisée = Nbrcde - Longueurliste))	1., 2. et 3.
5. ¬∃x (Stocker(x) ∨ Livrer(x)) → **X**Réserve = Réserve ∧ **X**Cderéalisée = Cderéalisée	ST25
6. Enregistrer(x) ∧ ¬∃y Stocker(y) → (Cderéalisée = Nbrcde - Longueurliste → **X**(Cderéalisée = Nbrcde - Longueurliste))	1., 3. et 5.

De ces différents cas, nous pouvons déduire :

7. Enregistrer(x) → (Cderéalisée = Nbrcde - Longueurliste → X(Cderéalisée = Nbrcde - Longueurliste))

Nous venons de démontrer dans les points b1) et b2) que si on avait l'invariant, celui-ci restait vrai lorsque "Livrer" ou "Enregistrer" s'exécutait. Pour démontrer b3) voyons ce qui se passe lorsque seule l'action "Stocker" s'exécute.

1. Stocker(x) → **X**Réserve = Réserve + x	ST19
2. ¬∃x (Enregistrer(x) ∨ Livrer(x)) → **X**Liste = Liste ∧ **X**Qcdée = Qcdée ∧ **X**Nbrcde = Nbrcde ∧ **X**Longueurliste = Longueurliste	ST16
3. Stocker(y) ∧ ¬∃x Enregistrer(x) ∧ ¬∃z Livrer(z) → (Cderéalisée = Nbrcde - Longueurliste → **X**(Cderéalisée = Nbrcde - Longueurliste))	1. et 2.

La démonstration de b.4) est triviale car, par l'axiome de localité, il apparaît que si aucune des actions de l'objet ne se réalise, aucun attribut ne change donc, a fortiori, les attributs composant l'invariant.

3.5 Implémentation d'objets abstraits

Ce que nous venons de voir dans les sections précédentes constitue la décomposition horizontale, c'est-à-dire que l'objet "SYSTEME" est la composition d'objets de base dont les actions ont la même granularité de changement que les actions de "SYSTEME". Même granularité de changement, veut dire qu'elles s'exécutent en respectant la même échelle de temps, leur temps d'exécution étant le même. Donc, pour toutes ces actions, si l'une d'elles s'exécute à l'instant t, l'exécution est terminée à l'instant t+1.

Nous allons maintenant nous intéresser à la décomposition verticale. Il s'agit de pouvoir représenter un objet ayant une certaine granularité de temps en termes d'un objet ayant une granularité de temps plus fine. C'est le cas lorsque la description d'un objet ne pourrait être implémentée telle quelle dans un système, celui-ci ne sachant implémenter de façon atomique les actions de l'objet. Il faut dès lors arriver à exprimer la spécification de cet objet en termes d'objets manipulables par le système.

Examinons initialement le rapport qui existe entre deux objets dont la granularité de temps est différente.

3.5.1 Différence de granularité

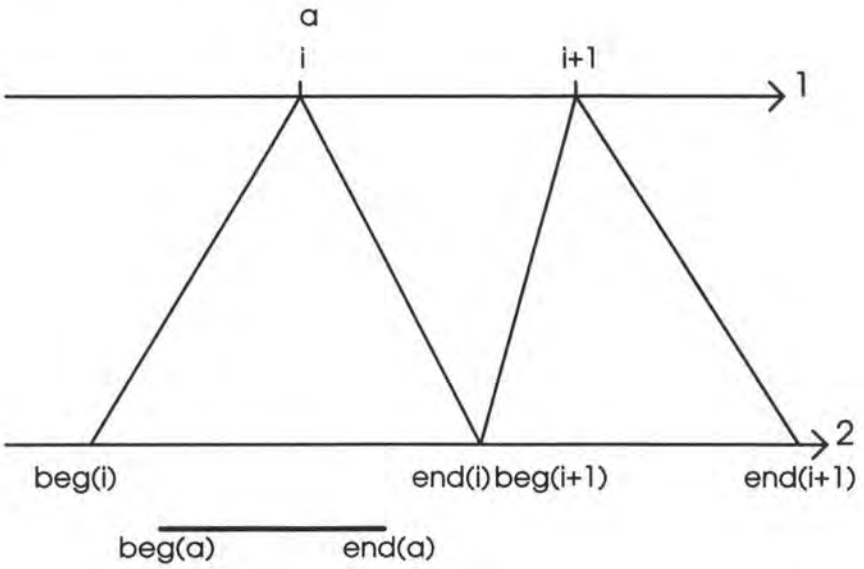

Fig. 3.5. : *Schéma exprimant la différence de granularité.*

Considérons deux axes, 1 et 2, représentant deux échelles de temps différentes, et voyons quelles sont les relations que nous pouvons établir entre des objets qui ont pour référence de temps ces deux axes. Prenons l'objet "X" possédant l'action "a" de granularité 1 et l'objet "Y" possédant les actions, de granularité 2, implémentant "a". La figure 3.5., nous montre que, si l'action "a" de "X" se produit à l'instant i, les effets de cette action sont réalisés à l'instant i+1, tandis que pour réaliser l'action "a" sur l'axe 2, l'objet "Y" doit réaliser plusieurs de ses actions pour atteindre le même but. L'instant i est donc éclaté en un intervalle de temps sur l'axe 2. Comme nous l'avons dit ci-dessus, certains objets ne peuvent être directement implémentés car le processeur concret utilisé ne peut exécuter les actions de ces objets de façon atomique. Dès lors nous allons devoir implémenter ces objets en termes d'objets concrets pouvant être utilisés par le processeur concret.

Note : une interprétation[17] ι de Σ_A est une extension Σ_A' de Σ_K. Cette interprétation peut aussi être étendue aux attributs de A. Une formule $\iota(\varphi)$ sera donc l'interprétation de la formule φ dans l'objet concret (φ est une formule du premier ordre sur les attributs de l'objet abstrait).

Prenons un exemple pour mieux comprendre comment implémenter un objet abstrait en terme d'un objet concret. Considérons un processeur concret qui ne peut faire que "plus un" et

[17]Ce problème a été développé dans [Veloso & al 85].

"moins un". Si on veut implémenter notre objet "STOCK" de l'exemple 3.1, on ne pourra implémenter ses actions "Approvisionner" et "Enlever" telles quelles car leur granularité ne correspond pas à celle de notre processeur. Il va dès lors falloir transformer cet objet en un objet implémentable.

Construisons un objet dont la granularité est identique à celle du processeur utilisé. Soit l'objet "CONCRET" possédant un attribut "Quantité" et possédant deux actions "Incrémenter" et "Décrémenter" qui permettent respectivement d'ajouter et de retirer une unité à "Quantité".

CONCRET

Sorts : NAT, BOOL

Symboles d'opération : succ,pred : NAT → NAT
> : NAT × NAT → BOOL
0 : → NAT

Symboles d'attribut : Quantité

Symboles d'action : Incrémenter
Décrémenter

Axiomes : Incrémenter → **X**Quantité = succ(Quantité)
Décrementer → **X**Quantité = pred(Quantité)
Décrementer → Quantité > 0
BEG → Quantité = 0

Le résultat de l'action "Approvisionner(3)" de l'objet "STOCK", sera le même que si l'on exécute trois fois l'action "Incrémenter" de l'objet "CONCRET". L'action abstraite "Approvisionner(3)" ne peut donc être réalisée en un instant au niveau du processeur concret. Il faut éclater l'instant abstrait en, au moins, trois instants concrets.

Reste à trouver une méthode qui nous permettra de traduire une spécification de granularité supérieure en une spécification de granularité inférieure lui correspondant. Voyons donc comment implémenter un objet abstrait en un objet concret.

3.5.2 Méthode d'implémentation

Implémenter une description d'objet abstrait sur un objet concret revient à définir une structure de conception pour l'objet concret. Elle sera composée de trois éléments : l'objet concret lui-même, qui sera le noyau de la structure de conception, un groupe d'objets que nous appellerons les interfaces et un objet particulier, le corps. L'implémentation se fait en deux phases :

1°) construire pour chaque action a_i de l'objet abstrait un objet interface "ITF_{ai}" (dont la description sera détaillée plus loin);

2°) faire la somme (dans le sens défini précédemment) de ces objets et de l'objet concret.

Le résultat de cette somme est le corps, traduction en termes de spécifications concrètes de l'objet abstrait.

Si on prend comme objet abstrait l'objet "A" et comme objet concret l'objet "N", l'implémentation peut être représentée par le schéma suivant :

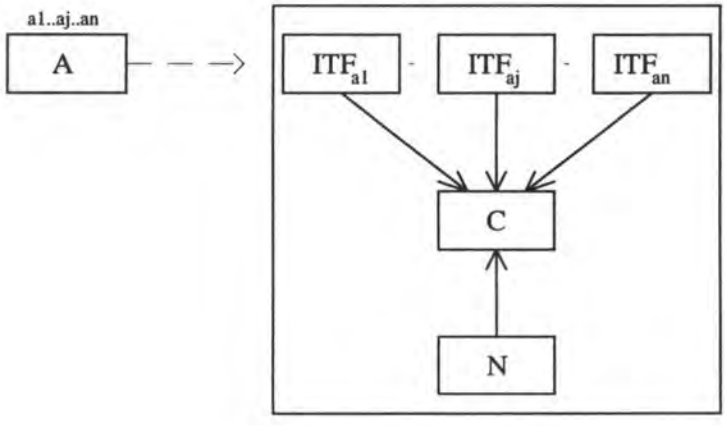

Fig.3.6. : *Implémentation de "A" sur "N".*

Où ITF_{a1} ... ITF_{an} sont les objets qui représentent les actions de A.

La structure de conception impliquée dans ce processus doit être d'un genre spécial : ses interfaces doivent être telles que nous pourrons abstraire de leur cycle de vie les occurrences des actions de plus haut niveau. L'idée est d'identifier dans chaque interface les points (actions

"concrètes") qui marquent le commencement et la fin de l'action qui a été abstraite à partir de l'interface[18].

Une structure de conception pour l'abstraction doit satisfaire les conditions suivantes :

o chaque interface ITF_a comprend deux actions distinctes beg_a et end_a, et un attribut in_a: BOOL. Ces éléments satisfont les conditions suivantes :

$ITF_a \Rightarrow (\mathbf{BEG} \rightarrow \neg in_a)$
$ITF_a \Rightarrow (\neg in_a \rightarrow ((\mathbf{X}\neg in_a) \mathcal{W} beg_a))$
$ITF_a \Rightarrow (beg_a \rightarrow \neg in_a \wedge (\mathbf{X} in_a \vee end_a))$
$ITF_a \Rightarrow (in_a \rightarrow ((\mathbf{X} in_a) \mathcal{W} end_a))$
$ITF_a \Rightarrow (end_a \rightarrow (in_a \vee beg_a) \wedge \mathbf{X}\neg in_a)$;

o le corps "C" est donc une extension du noyau "N" avec les actions d'interfaces et l'attribut in_a pour chaque interface ITF_a;

o les actions du noyau sont privées pour la structure de conception, pour chaque action n du noyau nous devons avoir comme théorème:

$$C \Rightarrow (n \rightarrow \bigvee_{i=1..n} \bigvee_{a \in Ai} in_a)$$

i.e. une action du noyau peut seulement se produire si elle est appelée par une des interfaces. Cette restriction implémentera (partiellement) la condition de localité de l'objet A.

3.5.3 Gestion de la concurrence

Réexaminons la question de l'éclatement des instants abstraits en intervalles d'instants concrets qui a été abordée au point 3.5.1. en affinant le problème de la gestion de la concurrence entre actions telle qu'elle a été exposée dans la théorie initiale.

Considérons trois actions "a", "b" et "c" d'un même objet abstrait "A". Pour mettre en évidence les problèmes qui peuvent survenir dans la concurrence, nous allons considérer qu'à l'instant abstrait i, "a" et "b" s'exécutent et à l'instant i+1, "c" s'exécute. La figure 3.7. nous

[18]De tels points de début et de fin ont été utilisés dans la littérature, mentionnés [Aceto & Hennessy 89] dans le contexte des algèbres de Processus.

montre une découpe possible d'intervalles concrets pour les trois actions. Pour que l'implémentation concrète respecte la spécification abstraite, certaines conditions devront être respectées au niveau concret.

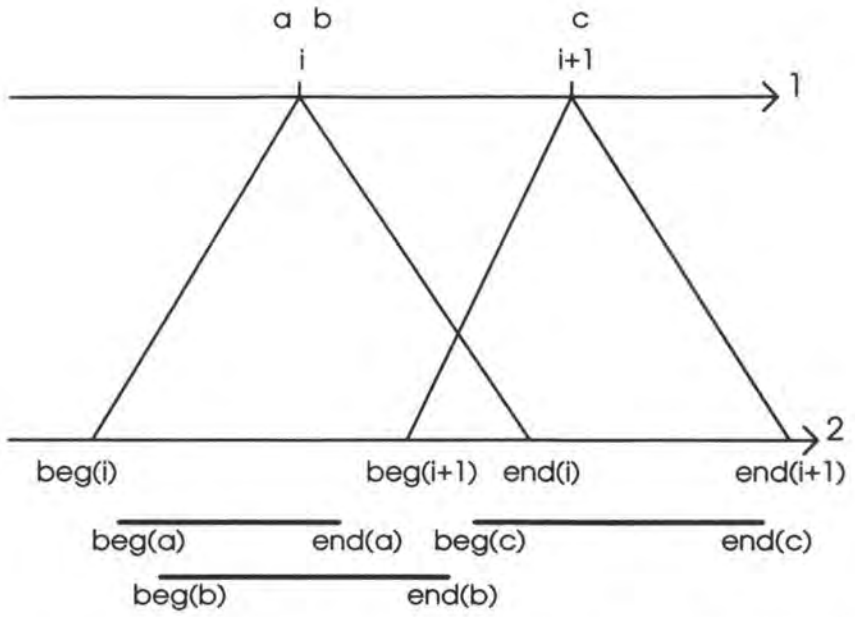

Fig.3.7. *: Eclatement des instants i et i+1 et des actions "a", "b" et "c".*

Dans notre exemple les actions "a" et "b" se réalisent au même instant abstrait, elles sont réalisées de façon atomique, donc elles s'exécutent exactement au même moment. Dans l'implémentation concrète, il faut tenir compte de ce fait. Ce qui implique que si beg(a) se produit avant beg(b), aucun attribut ne peut être modifié pendant l'intervalle [beg(a), beg(b)]. De même quand end(a) se réalise, plus aucun attribut ne peut être modifié jusqu'à la fin de l'intervalle [beg(i),end(i)]. Si ces conditions sont respectées, l'atomicité de "a" et de "b" est garantie. De manière générale, on peut dire que pour toute action "a" de l'intervalle i, la valeur des attributs à l'instant beg(a) doit être la même qu'à l'instant beg(i). De même, la valeur des attributs après le end(a) doit rester inchangée jusqu'à la survenance du end(i).

La théorie admet que deux intervalles se superposent, ce qui implique que les attributs doivent rester inchangés pendant cette intersection. Du fait de cette superposition, rien n'empêche que beg(c) se produise avant end(i) tant que les attributs restent inchangés durant l'intervalle [beg(i+1),end(i)]. Cette condition est déduite de ce qui a été dit supra. En effet, la valeur qu'ont les attributs lors de l'intervalle [beg(i),end(i)] ne peut avoir été modifiée que par l'implémentation des actions abstraites de i et non par l'implémentation des actions abstraites de i+1.

Remarque : la distinction entre intervalle concret d'exécution et exécution concrète à proprement parlé doit être faite. En effet, il se peut que lors de certains instants d'un intervalle concret, aucune action concrète ne s'exécute.

Formalisons les conditions précitées:
prenons une structure d'interprétation $I_A=(U_A,A_A,G_A)$ pour l'objet abstrait "A",
où l'on a:
⇨ $G_A : \Gamma_A^\bullet \rightarrow P_A(N_0)$ donc $\forall$ act $\in \Gamma_A^\bullet$ on a G_A(act) représente l'ensemble des instants auxquels l'action abstraite "act" de "A" se produit ;
et une structure d'interprétation $I_C=(U_C,A_C,G_C)$ pour le corps C,
où l'on a:
⇨ $A_C : A_C^\bullet \times N_0 \rightarrow s_U$ donc $\forall$ att $\in A_C^\bullet$ et i$\in N_0$ on a A_C(att)(i) représente la valeur de l'attribut att à l'instant i ;
⇨ $G_C : \Gamma_C^\bullet \rightarrow P_C(N_0)$ donc $\forall$ act $\in \Gamma_C^\bullet$ on a G_C(act) représente l'ensemble des instants auxquels l'action concrète "act" de "C" se produit.

Chaque abstraction consiste donc en :

1) une fonction $\rho : w \rightarrow 2^w$ (du temps abstrait vers le temps concret)
telle que pour chaque i $\in$ w,

a) ρ(i) est un intervalle [beg(i),end(i)] ;

b) beg(i) < beg(i+1) et end(i) < end(i+1) ;

c) pour tout m beg(i+1) $\leq$ m < end(i) et tout attribut f $\in A_N^\bullet$, $A_C(f)(m)= A_C(f)(m+1)$ (cette condition traduit le fait qu'aucun attribut ne change durant la superposition des intervalles);

2) la fonction $G_A : \Gamma_A^\bullet \rightarrow P_A(N_0)$ telle que, pour tout a $\in \Gamma_A^\bullet$ et i $\in G_A$(a), il existe exactement un j $\in \rho$(i) et un k $\in \rho$(i), j $\leq$ k tels que

a) j $\in G_C(beg_a)$;

b) k $\in G_C(end_a)$;

c) $A_C(f)(m)= A_C(f)(m+1)$ pour tout attribut f $\in A_N^\bullet$ et beg(i) $\leq$ m < j (aucune action ne peut commencer si les valeurs des attributs ont changé);

d) $A_C(f)(m)= A_C(f)(m+1)$ pour tout attribut f $\in A_N^\bullet$ et k < m $\leq$ end(i) (après la survenance du end de l'action, plus aucun attribut ne peut changer jusqu'à la fin de l'intervalle);

3) si $j \in G_C(act)$ pour une action "act" d'un interface ITF_a, alors il existe un $i \in w$ tel que $j \in \rho(i)$ et $i \in G_A(a)$. Autrement dit, les actions des interfaces ne peuvent s'exécuter qu'à l'intérieur des exécutions des actions abstraites.

Definition 3.5.3.1. (A-interprétation)
La A-interprétation ($A_{(\rho, G_A)} : A_A^{\bullet} \times N_0 \rightarrow s_U$, $G_{(\rho, G_A)} : \Gamma_A^{\bullet} \rightarrow P_A(N_0)$) abstraite à travers ($\rho, G_A$) est définie comme suit :

- $A_{(\rho, G_A)}(f)(i) = A_C(f)(beg(\rho(i)))$;
- $G_{(\rho, G_A)} = G_A$

Pour qu'un objet "C", soit l'implémentation d'un objet "A" il faut que pour toutes les interprétations de l'objet concret, la structure d'interprétation abstraite de la manière définie ci-dessus soit une structure d'interprétation de l'objet abstrait.

Chapitre 4

Développement d'un exemple

4.1. Introduction

Pour illustrer la théorie proposée par J.L. Fiadeiro et T. Maibaum, nous développons dans ce chapitre un exemple d'implémentation d'objet abstrait. Bien que celui-ci soit assez simple, il est représentatif des différentes situations que l'on peut rencontrer dans de la théorie.

Les deux premières sections, décriront les objets de l'exemple. La méthode d'implémentation des actions abstraites en termes d'actions concrètes est abordée infra. La quatrième section envisagera deux approches de construction du système final.

4.2. Description de l'objet abstrait

L'exemple développé ici est trivial et s'inspire de l'exemple 3.1. Nous allons considérer l'objet abstrait "A" qui est une simplification de l'objet "STOCK".

Sa description est :

A

Sorts : NAT, BOOL

Symboles d'opération : Fonctions et constantes habituelles pour les booléens.
+, - : NAT × NAT → NAT
≤ : NAT × NAT → BOOL
0 : → NAT

Symboles d'attribut : Quantité : NAT

Symboles d'action : Ajouter(NAT)
Soustraire(NAT)

Axiomes : A1) **BEG** → Quantité = 0
A2) Ajouter(x) → **X**Quantité = Quantité + x
A3) Soustraire(x) → **X**Quantité = Quantité - x
A4) Soustraire(x) → (x ≤ Quantité) = true

Au début de la période, l'attribut "Quantité" est initialisé à zéro. Les actions "Ajouter" ou "Soustraire" ont respectivement pour effet d'additionner et de soustraire un entier à cet attribut. L'action "Soustraire" ne peut être effectuée que si la valeur qui doit être soustraite est inférieure à la quantité restante.

4.3. Description de l'objet concret

Comme nous l'avons vu dans le chapitre 3, le processeur ne peut pas toujours implémenter certains objets. Les actions de ces objets doivent être décomposées en des actions plus simples dont l'exécution aura le même effet que ces actions abstraites.

Reprenons le processeur ne pouvant faire qu'additionner un et soustraire un à l'attribut "Quantité". Les actions "Ajouter" et "Soustraire" n'ont pas la granularité nécessaire et doivent être raffinées. Nous allons implémenter l'objet abstrait "A" sur un objet concret dont le but est d'additionner ou de soustraire un à une certaine quantité. La description du noyau de la structure de conception est identique à celle de l'objet "CONCRET" du Chapitre 3.

N

Sorts : NAT, BOOL

Symboles d'opération : Fonctions et constantes habituelles pour les booléens.
succ, pred : NAT → NAT
> : NAT × NAT → BOOL
0 : → NAT

Symboles d'attribut : Quantité : NAT

Symboles d'action : Incrémenter
Décrémenter

Axiomes : N1) **BEG** → Quantité = 0
N2) Incrémenter → **X**Quantité = succ(Quantité)
N3) Décrémenter → **X**Quantité = pred(Quantité)
N4) Décrémenter → Quantité > 0

Les actions "Incrémenter" et "Décrémenter" ont pour effet d'ajouter ou de soustraire un à l'attribut "Quantité" et on ne peut faire l'action "Décrémenter" que si "Quantité" est supérieure à zéro.

Pour implémenter l'objet "A" sur l'objet "N", nous devons construire à partir de "N" un système qui aura le même comportement que l'objet abstrait "A". Ce système ne pourra pas utiliser les actions "Ajouter" et "Soustraire" et devra donc les décomposer en termes des actions "Incrémenter" et "Décrémenter". Pour réaliser l'action "Ajouter(x)" par exemple, le système devra exécuter x fois l'action "Incrémenter".

4.4. Raffinement des actions sur des objets

Appliquons ce qui a été vu dans la théorie. Implémenter un objet "A" sur un objet "N" revient à définir une structure de conception pour "N".

La description des deux interfaces des actions "Ajouter" et "Soustraire" est la suivante:

$ITF_{Ajouter}$

Sorts : NAT, BOOL

Symboles d'opération : Fonctions et constantes habituelles pour les booléens.

Symboles d'attribut : $In_{Ajouter}(NAT) : BOOL$

Symboles d'action : $Beg_{Ajouter}(NAT)$
$End_{Ajouter}(NAT)$

Axiomes :
IA1) $\mathbf{BEG} \rightarrow \neg In_{Ajouter}(x)$
IA2) $\neg In_{Ajouter}(x) \rightarrow ((\mathbf{X}\neg In_{Ajouter}(x)) \; \mathcal{W} \; Beg_{Ajouter}(x))$
IA3) $Beg_{Ajouter}(x) \rightarrow \neg\exists y \; In_{Ajouter}(y) \wedge (\mathbf{X}In_{Ajouter}(x) \vee End_{Ajouter}(x))$
IA4) $In_{Ajouter}(x) \rightarrow ((\mathbf{X}In_{Ajouter}(x)) \; \mathcal{W} \; End_{Ajouter}(x))$
IA5) $End_{Ajouter}(x) \rightarrow (In_{Ajouter}(x) \vee Beg_{Ajouter}(x)) \wedge \mathbf{X}\neg In_{Ajouter}(x)$

$ITF_{Soustraire}$

Sorts : NAT, BOOL

Symboles d'opération : Fonctions et constantes habituelles pour les booléens.

Symboles d'attribut : $In_{Soustraire}(NAT) : BOOL$

Symboles d'action : $Beg_{Soustraire}(NAT)$
$End_{Soustraire}(NAT)$

Axiomes :
IS1) $\mathbf{BEG} \rightarrow \neg In_{Soustraire}(x)$
IS2) $\neg In_{Soustraire}(x) \rightarrow ((\mathbf{X}\neg In_{Soustraire}(x)) \; \mathcal{W} \; Beg_{Soustraire}(x))$
IS3) $Beg_{Soustraire}(x) \rightarrow \neg\exists y \; In_{Soustraire}(y) \wedge (\mathbf{X}In_{Soustraire}(x) \vee End_{Soustraire}(x))$
IS4) $In_{Soustraire}(x) \rightarrow ((\mathbf{X}In_{Soustraire}(x)) \; \mathcal{W} \; End_{Soustraire}(x))$
IS5) $End_{Soustraire}(x) \rightarrow (In_{Soustraire}(x) \vee Beg_{Soustraire}(x)) \wedge \mathbf{X}\neg In_{Soustraire}(x)$

Exécuter les actions abstraites "Ajouter" et "Soustraire" nécessite la réalisation d'une boucle sur les actions "Incrémenter" ou "Décrémenter". Il faut trouver un moyen pour formaliser cette boucle.
Elle pourrait être décrite en ajoutant une variable dont le but serait de compter le nombre d'itérations qu'il reste à faire. Celle-ci serait ajoutée dans le corps du système. Nous nous situons cependant dans une optique orientée objet, et dès lors, il est préferable d'introduire un nouvel objet qui gérera le comportement de cette variable.

Construisons l'objet "R".

R

Sorts : NAT, BOOL

Symboles d'opération : Fonctions et constantes habituelles pour les booléens.
prec : NAT → NAT
> : NAT × NAT → BOOL

Symboles d'attribut : Reste : NAT

Symboles d'action : Décroître
Initialiser(NAT)

Axiomes : R1) Initialiser(x) → **X**Reste = x
R2) Décroître → **X**Reste = prec(Reste)
R3) Décroître → (Reste > 0) = true
R4) (Reste > 0) = true → **F**Décroître

Intuitivement, il apparait que l'attribut "Reste" représente le nombre d'itérations restant à exécuter. "Reste" est initialisé et à chaque étape de la boucle, il est diminué de un. D'après R3, l'action "Décroître" ne s'exécutera que si "Reste" est supérieur à zéro. Autrement dit une itération ne se fera que si la boucle n'est pas terminée. L'axiome R4 guarantit la Vivacité de la boucle, c'est-à-dire que tant que la boucle n'est pas terminée, tôt ou tard, une itération s'exécutera.

4.5. Construction du Système

Les différents composants de la structure de conception étant créés, examinons les différentes méthodes qui contribuent à les relier. Nous analysons deux approches, l'une se basant sur la description axiomatique, l'autre utilisant la théorie des morphismes.

4.5.1. Première approche

4.5.1.1. Construction du corps

Pour implémenter l'objet "A" sur "N", nous avons vu qu'il était nécessaire d'avoir deux interfaces, une pour chaque action. Nous avons introduit aussi un nouvel objet pour décrire la boucle. Dès lors la structure générale décrite dans la théorie prend la forme suivante dans notre exemple.

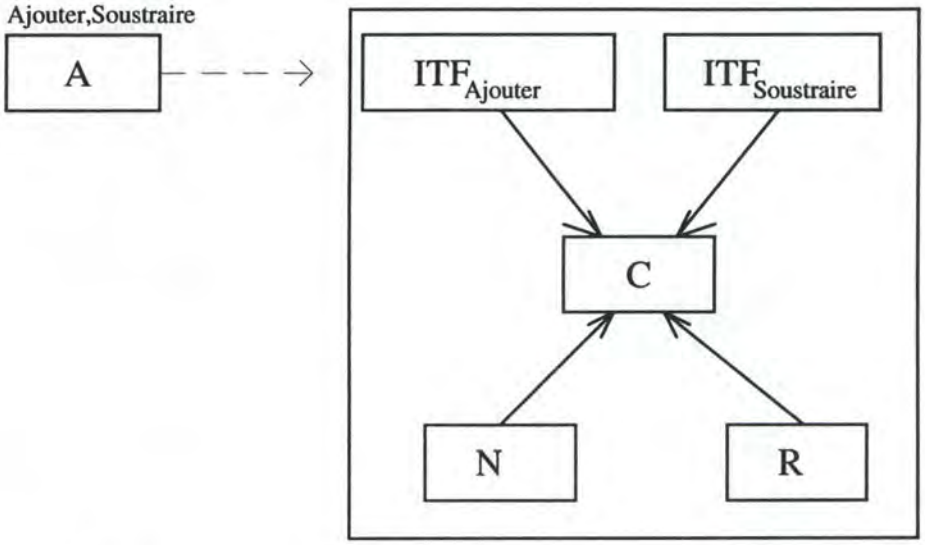

Fig. 4.1. *: Structure de conception pour "N" selon la première approche.*

Si le corps est strictement la somme des quatre composants, il n'aura pas le même comportement que "A". Des axiomes supplémentaires doivent être introduit pour que le comportement de l'objet "C" corresponde à celui de "A".

Lorsqu'on analyse la description de "A", on remarque que les actions "Ajouter" et "Soustraire" ne peuvent être exécutées en même temps (axiomes A2 et A3). Dans les axiomes

de "B", nous retrouverons des axiomes qui décrivent la boucle et des axiomes qui empêchent les actions "Ajouter" et "Soustraire" de s'exécuter simultanément.

C

Sorts : NAT, BOOL

Symboles d'opération : Fonctions et constantes habituelles pour les booléens.
prec, succ : NAT → NAT
>, ≤ : NAT × NAT → BOOL
0 : → NAT

Symboles d'attribut :
Quantité : NAT
Reste : NAT
$In_{Ajouter}$(NAT) : BOOL
$In_{Soustraire}$(NAT) : BOOL

Symboles d'action :
Décroître
Initialiser(NAT)
Incrémenter
Décrémenter
$Beg_{Ajouter}$(NAT)
$Beg_{Soustraire}$(NAT)
$End_{Ajouter}$(NAT)
$End_{Soustraire}$(NAT)

Axiomes :

{ Axiomes de "N". }

C1) Incrémenter → **X**Quantité = succ(Quantité)
C2) Décrémenter → **X**Quantité = prec(Quantité)
C3) Décrémenter → Quantité > 0
C4) **BEG** → Quantité = 0
C5) ¬(Incrémenter ∨ Décrémenter) → **X**Quantité = Quantité

{ Axiomes de "R". }

C6) Initialiser(x) → **X**Reste = x
C7) Décroître → **X**Reste = prec(Reste)
C8) Décroître → (Reste > 0) = true
C9) Reste > 0 → **F**Décroître
C10) ¬(Décroître ∨ Initialiser(x)) → **X**Reste = Reste

{ Axiomes de "$ITF_{Ajouter}$". }

C11) **BEG** → $\neg In_{Ajouter}(x)$
C12) $\neg In_{Ajouter}(x) \rightarrow ((\mathbf{X}\neg In_{Ajouter}(x))\ \mathcal{W}\ Beg_{Ajouter}(x))$
C13) $Beg_{Ajouter}(x) \rightarrow \neg\exists y\ In_{Ajouter}(y) \wedge (\mathbf{X}In_{Ajouter}(x) \vee End_{Ajouter}(x))$
C14) $In_{Ajouter}(x) \rightarrow ((\mathbf{X}In_{Ajouter}(x))\ \mathcal{W}\ End_{Ajouter}(x))$
C15) $End_{Ajouter}(x) \rightarrow (In_{Ajouter}(x) \vee Beg_{Ajouter}(x)) \wedge \mathbf{X}\neg In_{Ajouter}(x)$
C16) $\neg\exists x(Beg_{Ajouter}(x) \vee End_{Ajouter}(x)) \rightarrow \mathbf{X}In_{Ajouter}(x) = In_{Ajouter}(x)$

{ Axiomes de "$ITF_{Soustraire}$". }

C17) **BEG** $\rightarrow \neg In_{Soustraire}(x)$
C18) $\neg In_{Soustraire}(x) \rightarrow ((\mathbf{X}\neg In_{Soustraire}(x))\ \mathcal{W}\ Beg_{Soustraire}(x))$
C19) $Beg_{Soustraire}(x) \rightarrow \neg\exists y\ In_{Soustraire}(y) \wedge (\mathbf{X}In_{Soustraire}(x) \vee End_{Soustraire}(x))$
C20) $In_{Soustraire}(x) \rightarrow ((\mathbf{X}In_{Soustraire}(x))\ \mathcal{W}\ End_{Soustraire}(x))$
C21) $End_{Soustraire}(x) \rightarrow (In_{Soustraire}(x) \vee Beg_{Soustraire}(x)) \wedge \mathbf{X}\neg In_{Soustraire}(x)$
C22) $\neg\exists x\ (Beg_{Soustraire}(x) \vee End_{Soustraire}(x)) \rightarrow \mathbf{X}In_{Soustraire}(x) = In_{Soustraire}(x)$

{ L'axiome suivant implémente la condition de localité de "A". Il empêche les actions "Incrémenter" et "Décrémenter" de s'exécuter si on ne se trouve pas à l'intérieur des actions "Ajouter" et "Soustraire". L'attribut "Quantité" ne sera donc pas modifié si on ne se trouve pas à l'intérieur d'une de ces deux actions, ce qui est bien la condition de localité de "A". }

C23) Incrémenter $\vee$ Décrémenter $\rightarrow \exists x\ (In_{Ajouter}(x) \vee In_{Soustraire}(x))$

{ Description de la boucle. }

{ Initialisation : Lorsqu'une action ("Ajouter" ou "Soustraire") commence, le reste est initialisé. }

C24) $Beg_{Ajouter}(x) \rightarrow$ Initialiser(x)
C25) $Beg_{Soustraire}(x) \rightarrow$ Initialiser(x)
C26) Initialiser(x) $\rightarrow Beg_{Ajouter}(x) \vee Beg_{Soustraire}(x)$

{ Itération : Avec les axiomes suivants, nous indiquons que si l'action "Ajouter" ("Soustraire") est en train de se réaliser, "Incrémenter" ("Décrémenter") est synchronisé avec "Décroître". }

C27) Incrémenter $\rightarrow$ Décroître
C28) Décrémenter $\rightarrow$ Décroître
C29) Décroître $\rightarrow$ Incrémenter $\vee$ Décrémenter

{ Fin de la boucle : Si on est en train d'exécuter l'action "Ajouter" ("Soustraire") et si l'attribut "Reste" est égal à zéro, tôt ou tard, on terminera l'action mais elle ne se terminera qu'à cette condition. }

C30) $In_{Ajouter}(x) \wedge (Reste = 0) \rightarrow \mathbf{F}End_{Ajouter}(x)$
C31) $In_{Soustraire}(x) \wedge (Reste = 0) \rightarrow \mathbf{F}End_{Soustraire}(x)$
C32) $End_{Ajouter}(x) \rightarrow (Reste = 0 \wedge In_{Ajouter}(x))$
C33) $End_{Soustraire}(x) \rightarrow (Reste = 0 \wedge In_{Soustraire}(x))$

{ Les axiomes suivants indiquent que les actions "Soustraire" et "Ajouter" ne peuvent être exécutées en même temps. }

C34) $Beg_{Soustraire}(x) \rightarrow \neg\exists y\ In_{Ajouter}(y)$
C35) $Beg_{Ajouter}(x) \rightarrow \neg\exists y\ In_{Soustraire}(y)$
C36) Décrémenter $\rightarrow \neg\exists x\ In_{Ajouter}(x)$
C37) Incrémenter $\rightarrow \neg\exists x\ In_{Soustraire}(x)$

{ Axiome de Sécurité. }

C38) $Beg_{Soustraire}(x) \rightarrow x \leq Quantité$

A partir de l'exemple, déduisons la forme générale d'une structure de conception, si l'objet abstrait réclame l'utilisation d'une boucle. Nous y retrouvons trois types d'objets : les interfaces, les objets compteurs et l'objet concret. Ces trois types d'objets sont sommés pour former le corps.

Le corps est constitué de la somme des signatures (vu dans l'exemple 3.1. du chapitre 3) et d'un certain nombre d'axiomes. Classons ces axiomes en trois familles :

1) les axiomes se rapportant aux objets concrets de base. Cette famille peut être décomposée en deux sous-familles:

a) Les axiomes propres aux objets composant la somme (les interfaces, l'objet concret, les objets compteurs).

Ces axiomes sont de la forme des axiomes de Vivacité, de Sécurité, de Changement, de Début et de Synchronisation;

b) les axiomes exprimant la condition de localité de chacun des objets composants. Leur forme sera :

∀ objet composant "O" on a

$$\neg(\bigvee_{a\in\Gamma^{\bullet}_{o}} \exists x_a\ a(x_a)) \rightarrow (\bigwedge_{att\in A^{\bullet}_{o}} \forall x_{att}\ Xatt(x_{att}) = att(x_{att}))^{19}$$

2) les axiomes qui expriment certaines conditions pour respecter le comportement de l'objet abstrait. Ces axiomes sont classés en quatre sous-familles:

a) les axiomes qui expriment la condition de localité de l'objet abstrait.

Prenons pour objet abstrait "A" et pour objet noyau "N". Pour toutes actions $a \in \Gamma^{\bullet}_{A}$ nous définissons un sous-ensemble de $\Gamma^{\bullet}_{N}$ que nous notons $\Gamma^{\bullet}_{N_a}$, l'ensemble des actions de "N" qui participent à l'implémentation de "a". En utilisant ce nouvel ensemble, les axiomes de ce groupe ont la forme :

$$(\bigvee_{n\in\Gamma^{\bullet}_{N}} \exists x_n\ n(x_n)) \rightarrow (\bigvee_{a\in\Gamma^{\bullet}_{A}} \exists x_a\ In_a(x_a))$$

---

[19] pour chaque symbole y, x_y est un vecteur de variables, toutes distinctes, des sorts appropriés, cette note est valable pour toutes les formules décrites dans les sous-familles.

b) les axiomes qui expriment la condition de Sécurité de l'objet abstrait. Ces axiomes ont la forme:

$\forall a \in \Gamma_A^\bullet$, si on a l'axiome de Sécurité au niveau abstrait

$a(x_a) \rightarrow$ condition ,

devient au niveau concret

$\mathbf{Beg_a(x_a)} \rightarrow \iota\mathbf{(condition)}$

où ι(condition) est la traduction dans l'objet concret de la condition;

c) les axiomes qui expriment la condition de Vivacité de l'objet abstrait. Ils auront la forme :

$\forall a \in \Gamma_A^\bullet$ si on a l'axiome de Vivacité au niveau abstrait

condition $\rightarrow \exists x_a \, Fa(x_a)$,

il devient au niveau concret

$\iota\mathbf{(condition)} \rightarrow \exists \mathbf{x_a} \, \mathbf{FBeg_a(x_a)}$;

d) les axiomes qui traduisent, au niveau concret, le fait que certaines actions abstraites ne peuvent se réaliser en même temps.

$\forall$ a, b $\in \Gamma^\bullet_A$ et $\forall$ u, v $\in \Gamma^\bullet_N$ tels que u participe à l'implémentation de a et v à l'implémentation de b, et u, v entrent en conflit d'écriture ou de lecture-écriture au niveau de leurs attributs. Ces axiomes auront la forme :

$$\mathbf{Beg_a(x_a)} \rightarrow \neg\exists \mathbf{x_b} \, (\mathbf{In_b(x_b)} \vee \mathbf{Beg_b(x_b)})$$
$$\mathbf{Beg_b(x_b)} \rightarrow \neg\exists \mathbf{x_a} \, (\mathbf{In_a(x_a)} \vee \mathbf{Beg_a(x_a)})$$
$$\mathbf{u(x_u)} \rightarrow \neg\exists \mathbf{x_b} \, (\mathbf{In_b(x_b)} \vee \mathbf{Beg_b(x_b)})$$
$$\mathbf{v(x_v)} \rightarrow \neg\exists \mathbf{x_a} \, (\mathbf{In_a(x_a)} \vee \mathbf{Beg_a(x_a)})$$

3) les axiomes qui décrivent la boucle. Ils sont rassemblés en trois sous-familles:

a) les axiomes qui décrivent les initialisations à faire au début de l'exécution d'une action abstraite, autrement dit ce que chaque action beg des interfaces implique. La forme des axiomes est donc :

$$\bigvee_{a \in \Gamma_A^\bullet} \mathbf{Beg_a(x_a)} \rightarrow \mathbf{initialisation(x_a)}$$

$$\mathbf{initialisation(x_a)} \rightarrow \bigvee_{a \in \Gamma_A^\bullet} \mathbf{Beg_a(x_a)}$$

b) les axiomes qui spécifient quelles actions sont synchronisées avec l'action "décroître" du compteur.

Si, comme dans notre exemple, nous utilisons le même compteur pour plusieurs boucles, les axiomes auront la forme suivante :

b1 .. bn $\in \Gamma^{\bullet}_{N}$:

$\mathbf{b1}(x_{b1}) \rightarrow \exists x_{\text{Décroitre}}\ \mathbf{Décroître}(x_{\text{Décroitre}})$

...

$\mathbf{bn}(x_{bn}) \rightarrow \exists x_{\text{Décroitre}}\ \mathbf{Décroître}(x_{\text{Décroitre}})$

$\mathbf{Décroitre}(x_{\text{Décroitre}}) \rightarrow \exists x_{b1}\ \mathbf{b1}(x_{b1}) \vee ... \vee \exists x_{bn}\ \mathbf{bn}(x_{bn})$

où b1, ..., bn sont les dernières actions à être exécutées à l'intérieur de chacunes des n boucles à utiliser le compteur;

c) les axiomes qui décrivent les conditions de fin. Ceux-ci assurent que l'action se termine lorsque la condition de fin de boucle est vérifiée et uniquement une fois cette condition remplie.

$\forall$ a$\in \Gamma_{A}^{\bullet}$:

$\mathbf{In}_{a}(x_{a}) \wedge \mathbf{compteur} = 0 \rightarrow \mathbf{FEnd}_{a}(x_{a})$

$\mathbf{End}_{a}(x_{a}) \rightarrow \mathbf{In}_{a}(x_{a}) \wedge \mathbf{compteur} = 0$

Remarque: nous avons identifié trois familles principales et neuf sous familles. Il se peut que certains corps n'aient aucun axiome dans une des familles ou des sous-familles. En effet, dans notre exemple, "C" ne contient pas d'axiomes de Vivacité pour "A".

4.5.1.2. Démonstration des propriétés

Une fois le corps construit, il nous reste à démontrer que son comportement correspond bien à celui de l'objet abstrait "A". Pour cela on va prouver à l'aide des règles de transition que les propriétés vérifiées dans "A" le sont aussi dans "C".

Règles de transition

Pour réaliser les preuves, nous allons utiliser les lois de Changement, de Sécurité, de Vivacité et de Début.

Notation :

- "a" est une action de l'objet abstrait;
- les conditions φ et ψ sont des formules du premier ordre sur les attributs de l'objet A;
- $\iota(\varphi)$ est la traduction de la formule φ dans l'objet C.

1• Changement : $\dfrac{C \Rightarrow (Beg_a \wedge \iota(\varphi) \rightarrow \mathbf{F}(End_a \wedge \mathbf{X}\iota(\psi)))}{A \Rightarrow (a \wedge \varphi \rightarrow \mathbf{X}\psi)}$

2• Sécurité : $\dfrac{C \Rightarrow (Beg_a \rightarrow \iota(\varphi))}{A \Rightarrow (a \rightarrow \varphi)}$

3• Vivacité : $\dfrac{C \Rightarrow (\iota(\varphi) \rightarrow \mathbf{F}beg_a)}{A \Rightarrow (\varphi \rightarrow \mathbf{F}a)}$

4• Début : $\dfrac{C \Rightarrow (\mathbf{BEG} \rightarrow \iota(\varphi))}{A \Rightarrow (\mathbf{BEG} \rightarrow \varphi)}$

Cohérence des règles de transition

Avant de pouvoir utiliser ces règles de transition, nous devons nous assurer de leur cohérence. Nous allons donc établir la correction de ces formules. Les conditions vues au point 3.5.3. sont nécessaires à la démonstration.

1• Changement.

Remarque : De manière intuitive, pour que la règle soit cohérente, il faut avoir dans l'objet concret que si une condition commence avec une condition φ, elle se terminera tôt ou tard et, à l'instant suivant, la condition ψ sera vérifiée. Or, ce n'est pas exactement ce qui est dit dans la règle proposée dans la théorie. Nous avons $(Beg_a \wedge \iota(\varphi) \rightarrow \mathbf{F}(End_a \wedge \mathbf{X}\iota(\psi)))$. Le End_a dont on parle dans la deuxième partie n'est pas forcément celui qui correspond au Beg_a de la première partie. En effet, supposons que l'action "a" s'exécute au instant i et j avec $j > i$, nous aurons la fig.4.2

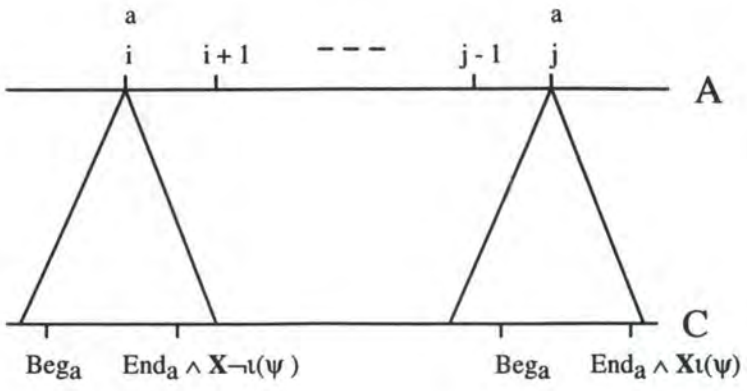

Fig 4.2. *: Eclatement de deux instants comprenant une occurence de "a"*

Intuitivement, nous voyons que nous pouvons avoir $C \Rightarrow (Beg_a \wedge \iota(\varphi) \rightarrow \mathbf{F}(End_a \wedge \mathbf{X}\iota(\psi)))$ sans que cela implique $A \Rightarrow (a \wedge \varphi \rightarrow \mathbf{X}\psi)$.

Pour remédier à ce problème, nous allons utiliser la règle de transition suivante :

1• Changement : $$\frac{C \Rightarrow (Beg_a \wedge \iota(\varphi) \rightarrow \mathbf{X}In_a\, \mathcal{U}\, (End_a \wedge \mathbf{X}\iota(\psi)))}{A \Rightarrow (a \wedge \varphi \rightarrow \mathbf{X}\psi)}$$

D'après la définition de l'opérateur temporel $\mathcal{U}$, nous $In_a\, \mathcal{U}\, (End_a \wedge \mathbf{X}\iota(\psi))$ est équivalent à $In_a\, \mathcal{W}\, (End_a \wedge \mathbf{X}\iota(\psi)) \wedge \mathbf{F}(End_a \wedge \mathbf{X}\iota(\psi))$. Dans notre nouvelle règle, nous avons juste ajouter que nous devions avoir $In_a\, \mathcal{W}\, (End_a \wedge \mathbf{X}\iota(\psi))$, autrement dit, nous avons préciser que le End_a de la deuxième partie correspond bien au Beg_a de la première partie. C'est bien ce que nous désirions faire.

1. La condition 2a) sur l'abstraction énonce que $\forall a \in \Gamma_A{}^\bullet$ et $i \in G_A(a)$, il existe exactement un $j \in \rho(i)$ tel que $j \in G_C(beg_a)$. Autrement dit, si "a" se produit à un instant abstrait i, l'action "beg_a" s'exécute durant l'intervalle $\rho(i)$.

2. D'après la condition 2c), nous savons que les attributs ont les mêmes valeurs au début de l'action qu'au début de l'intervalle, c'est à dire à l'instant $beg(\rho(i))$. Nous avons donc :
   si $j \in \rho(i)$ et $j \in G(beg_a)$
   $A_C(\iota(\varphi))(j) = A_C(\iota(\varphi))(beg(\rho(i)))$.
   De la définition de l'A-interprétation, nous savons que
   $A_{(\rho, G_A)}(\varphi)(i) = A_C(\iota(\varphi))(beg(\rho(i)))$.

En effet, φ est une formule du premier ordre sur les attributs de l'objet abstrait.
Par transition, nous obtenons :
$A_C(\iota(\varphi))(j) = A_{(\rho, G_A)}(\varphi)(i)$.
Si nous avons la formule φ à l'instant i où "a" s'exécute, nous aurons ι(φ) lorsque "beg_a" s'exécutera durant l'intervalle ρ(i).

3. D'après la définition de l'opérateur temporel **X**, pour avoir **X**φ à un instant i, il faut avoir φ à l'instant i + 1. De la définition de l'A-interprétation, nous avons :
$A_{(\rho, G_A)}(\varphi)(i + 1) = A_C(\iota(\varphi))(beg(\rho(i + 1)))$.
Nous allons donc démontrer que $A_C(\iota(\varphi))(beg(\rho(i + 1))) = A_C(\iota(\varphi))(j + 1)$ avec $j \in \rho(i)$ et $j \in G(end_a)$.
Nous avons vu dans le chapitre 3 que deux intervalles pouvaient éventuellement se superposer, envisageons les deux cas séparément.

A. Intervalles disjoints.

Lemme 4.5.1.2.1.: les attributs du noyau ne changent pas dans l'intervalle]end(i),beg(i+1)[.
Comme les actions du noyau sont privées, nous avons :
$C \Rightarrow (act \rightarrow \bigvee_{i=1..n} \bigvee_{a \in \Gamma\Gamma F_{ai}} a)$
Une action du noyau ne peut se produire que si elle est appelée par une action de l'interface. Or, d'après la condition 3), une action de l'interface ne peut se dérouler qu'à l'intérieur d'un intervalle. Une action du noyau ne peut donc pas s'exécuter à un instant qui n'appartient pas à un intervalle. En nous basant sur la condition de localité du noyau nous pouvons dire que les attributs de celui-ci ne pourront pas être modifiés si nous ne nous trouvons pas dans un intervalle. Comme φ est une formule du premier ordre sur les attributs de l'objet abstrait et vu que ceux-ci ont leur traduction dans les attributs du noyau, la formule ι(φ) ne pourra pas être modifiée si nous ne sommes pas à l'intérieur d'un intervalle.
La condition 2d) nous assure qu'après l'exécution de end_a, les valeurs des attributs restent inchangées jusqu'à la fin de l'intervalle. Nous pouvons dire que :
$A_C(\iota(\varphi))(j + 1) = A_C(\iota(\varphi))(end(\rho(i))$ avec $j \in \rho(i)$ et $j \in G(end_a)$.
D'après le lemme 4.5.1.2.1., nous avons $A_C(\iota(\varphi))(beg(\rho(i + 1)) = A_C(\iota(\varphi))(end(\rho(i))$
Par transition :
$A_C(\iota(\varphi))(beg(\rho(i + 1)) = A_C(\iota(\varphi))(j + 1)$

B. Intervalles superposés.

D'après la condition 1c), aucun attribut du noyau ne peut être modifié dans l'intervalle.
$A_C(\iota(\varphi))(m) = A_C(\iota(\varphi))(m + 1)$ pour tout $beg(\rho(i + 1)) \leq m < end(\rho(i))$.
Nous aurons donc $A_C(\iota(\varphi))(beg(\rho(i + 1)) = A_C(\iota(\varphi))(end(\rho(i)))$.
La condition 2d) nous dit que :
$A_C(\iota(\varphi))(j + 1) = A_C(\iota(\varphi))(end(\rho(i))$ avec $j \in \rho(i)$ et $j \in G(end_a)$.
Par transition, nous pouvons déduire que :
$A_C(\iota(\varphi))(beg(\rho(i + 1)) = A_C(\iota(\varphi))(j + 1)$.

Des points 1., 2., 3. et $C \Rightarrow (Beg_a \wedge \iota(\varphi) \rightarrow \mathbf{F}(End_a \wedge \mathbf{X}\iota(\psi)))$, nous pouvons déduire la règle de changement. En effet, si $A \Rightarrow a \wedge \varphi$ est vérifié, $C \Rightarrow Beg_a \wedge \iota(\varphi)$ l'est aussi (points 1. et 2.). Si $C \Rightarrow \mathbf{F}(End_a \wedge \mathbf{X}\iota(\psi))$ est vérifié, $A \Rightarrow \psi$ l'est aussi (point 3). De $C \Rightarrow (Beg_a \wedge \iota(\varphi) \rightarrow \mathbf{F}(End_a \wedge \mathbf{X}\iota(\psi)))$ nous pouvons donc déduire $A \Rightarrow (a \wedge \varphi \rightarrow \mathbf{X}\psi)$.

2• Sécurité

Pour cette démonstration nous nous basons sur les points de la démonstration précédente. Si $A \Rightarrow a$ est vérifié, $C \Rightarrow Beg_a$ l'est aussi (Point 1. de la démonstration pour la règle de changement). Si $C \Rightarrow \iota(\varphi)$ est vérifié, $A \Rightarrow \varphi$ l'est aussi (Point 2. de la démonstration pour la règle de changement). De $C \Rightarrow (Beg_a \rightarrow \iota(\varphi))$, nous pouvons donc déduire $A \Rightarrow (a \rightarrow \varphi)$.

3• Vivacité

D'après la définition de l'opérateur **F**, **F**act est vrai à un instant i si et seulement si l'action "act" s'exécute à un instant j avec $j \geq i$.

1. Si nous avons φ à un instant abstrait i, d'après la définition de l'A-interprétation, nous avons $A_{(\rho, G_A)}(\varphi)(i) = A_C(\iota(\varphi))(beg(\rho(i))$.
2. La condition 3) sur l'abstraction nous dit que si à un instant concret j, l'action "beg_a" s'exécute, il existe i tel que $j \in \rho(i)$ et $i \in G_A(a)$.

Si $A \Rightarrow \varphi$ est vérifié, $C \Rightarrow \iota(\varphi)$ l'est aussi (Point 1.). Par la définition de l'opérateur temporel **F**, $C \Rightarrow \mathbf{F}beg_a$ est vérifié à l'instant j si beg_a est exécuté à un instant k avec $k \geq j$. Si $j \in \rho(i)$ et $k \in \rho(l)$, nous avons $l \geq k$. Si $C \Rightarrow \mathbf{F}beg_a$ est vérifié, $A \Rightarrow \mathbf{F}a$ l'est aussi.
De $C \Rightarrow (\iota(\varphi) \rightarrow \mathbf{F}beg_a$ nous pouvons donc déduire $A \Rightarrow (\varphi \rightarrow \mathbf{F}a)$.

4• Début

Pour démontrer la cohérence de cette règle, nous posons l'hypothèse que le début de la période concrète correspond au début de la période abstraite. Autrement dit, $beg(\rho(0)) = 0$.

1. D'après l'hypothèse que nous avons fait ci-dessus, nous avons que si A $\Rightarrow$ **BEG**, nous avons C $\Rightarrow$ **BEG**.
2. D'après la définition de l'A-abstraction, nous avons $A_{(\rho, G_A)}(\varphi)(0) = A_c(\iota(\varphi))(beg(\rho(0)) = A_c(\iota(\varphi))(0)$

De ces deux points, nous pouvons déduire la règle de Début.

Propriétés à démontrer

Les propriétés qui doivent être vérifiées dans C sont :

1. ***BEG*** $\rightarrow$ *Quantité = 0.*
2. *Soustraire(x)* $\rightarrow$ *(x* $\leq$ *Quantité) = true*
3. *Ajouter(x)* $\rightarrow$ ***X****Quantité = Quantité + x*
4. *Soustraire(x)* $\rightarrow$ ***X****Quantité = Quantité - x*

La propriété 1. se déduit de la loi 4• et de l'axiome B4.
La propriété 2. se déduit de la loi 2• et de l'axiome C38.

La démonstration des propriétés 3. et 4. ayant la même forme, nous nous contenterons de prouver la propriété 3.

Que devons nous prouver ?

Dans l'objet "A", nous avons :

$$Ajouter(x) \rightarrow \ XQuantité = Quantité + x$$

Que nous transformons en :

$$Ajouter(x) \wedge \ Quantité = Quantité_0 \rightarrow \ XQuantité_0 = Quantité + x$$

D'après la règle 1• nous devons démontrer que nous avons dans l'objet "C" :

$$Beg_{Ajouter}(x) \wedge \ Quantité = Quantité_0 \rightarrow \ XIn_{Ajouter}(x) \ \mathcal{U} \ (End_{Ajouter}(x) \wedge X(Quantité = Quantité_0 + x))$$

Comment le prouver ?

D'après la définition de l'opérateur temporel $\mathcal{U}$, nous devons prouver deux choses :

1. $Beg_{Ajouter}(x) \wedge Quantité = Quantité_0 \rightarrow \mathbf{F}(End_{Ajouter}(x) \wedge \mathbf{X}(Quantité = Quantité_0 + x))$
2. $Beg_{Ajouter}(x) \wedge Quantité = Quantité_0 \rightarrow \mathbf{X}In_{Ajouter}(x)\ \mathcal{W}\ (End_{Ajouter}(x) \wedge \mathbf{X}(Quantité = Quantité_0 + x))$

Méthode de preuve pour le point 1

Pour prouver la correction d'un programme comprenant une boucle par la méthode de l'invariant[20], il faut démontrer la correction partielle et la terminaison du programme.

La correction partielle

Notation : P, Q : deux assertions sur les variables du programme.
S : une partie du programme.
{P} S {Q}signifiera "Si l'exécution de S est déclenchée
avec des valeurs de variables qui vérifient P, l'exécution de S se terminera
sans erreur et les valeurs finales des variables vérifieront l'assertion Q".

Pour démontrer la correction partielle, il faut trouver un invariant tel qu'on puisse prouver les trois points suivants :

a. {Précondition} Initialisation {Invariant};
b. {Invariant et non condition de sortie de la boucle} Itération {Invariant};
c. {Invariant et condition de sortie de la boucle} Cloture {Postcondition}.

La terminaison

D'après la correction partielle, nous savons que si le programme se termine, il se terminera sans erreur et les spécifications seront respectées. Il nous reste donc à démontrer que le programme se terminera. Pour cela, il faut montrer que la condition de sortie de boucle se réalisera obligatoirement.

---

[20]Méthode reprise du cours "*Preuves de programmes*" de B. Lecharlier [Lecharlier 91].

La méthode est la suivante :
- trouver une fonction H, qui peut dépendre de toutes les variables du programme;
- montrer qu'elle est bornée supérieurement (inférieurement) lors de chaque passage dans la boucle;
- établir que H croît (décroît) strictement à chaque itération.

Application de la méthode de l'invariant à notre théorie

Pour pouvoir appliquer cette méthode, nous devons donner l'équivalent dans notre théorie des différentes parties de programmes citées ci-dessus.

Correction partielle

Dans notre approche, l'initialisation sera l'ensemble des actions exécutées lors du passage de l'instant i à l'instant i + 1 lorsqu'on a Beg_{action} à l'instant i.
L'itération sera l'ensemble des actions exécutées lors du passage de l'instant i à l'instant i + 1 lorsqu'on a In_{action} à l'instant i.
La cloture sera l'ensemble des actions exécutées lors du passage de l'instant i à l'instant i + 1 lorsqu'on a End_{action} à l'instant i.

En résumé, il faut trouver un invariant qui respecte les trois points suivants :

a. $Beg_{action} \wedge$ Précondition $\rightarrow$ **X**Invariant;
b. $In_{action} \wedge$ Invariant $\rightarrow$ **X**Invariant;
c. $End_{action} \wedge$ **X**Invariant $\rightarrow$ **X**Postcondition;

où Précondition et Postcondition sont des formules du premier ordre sur les attributs du noyau.

Grâce aux axiomes de l'interface de "action", nous démontrons que si nous parvenons à trouver un invariant qui respecte ces trois points, nous aurons prouvé la correction partielle (si l'action débute avec la Précondition et si elle se termine, alors, elle se termine avec la Postcondition).

1. $Beg_{action} \rightarrow \neg In_{action} \wedge (XIn_{action} \vee End_{action})$ — Axiome de l'interface
2. $Beg_{action} \rightarrow XIn_{action} \vee End_{action}$ — 1. et P.C.[21]

---

[21] Par souci de simplification, nous notons P.C. lorsqu' une étape qui peut être justifiée par une règle du calcul propositionnel.

Deux cas peuvent se produire :

A. End_{action} se réalise

3. $Beg_{action} \rightarrow End_{action}$	2., P.C. et hyp. A.
4. $Beg_{action} \wedge$ Précondition $\rightarrow End_{action}$	3. et P.C.
5. $Beg_{action} \wedge$ Précondition $\rightarrow$ **X**Invariant	a.
6. $Beg_{action} \wedge$ Précondition $\rightarrow End_{action} \wedge$ **X**Invariant	4. et 5.
7. $End_{action} \wedge$ **X**Invariant $\rightarrow$ **X**Postcondition	c.
8. $Beg_{action} \wedge$ Précondition $\rightarrow$ **X**Postcondition	6. et 7.

B. End_{action} ne se réalise pas

9. $Beg_{action} \rightarrow$ **X**In_{action}	2., P.C. et hyp B.
10. $Beg_{action} \wedge$ Précondition $\rightarrow$ **X**In_{action}	9. et P.C.
11. $Beg_{action} \wedge$ Précondition $\rightarrow$ **X**($In_{action} \wedge$ Invariant)	a., 10. et P.C.
12. $In_{action} \wedge$ Invariant $\rightarrow$ **X**Invariant	b.
13. $In_{action} \rightarrow$ **X**In_{action} $\mathcal{W}$ End_{action}	Axiome de l'interface

De nouveau, deux cas peuvent se produire.

B1. End_{action} se réalise

14. $In_{action} \rightarrow End_{action}$	13., P.C. et hyp. B1.
15. $In_{action} \wedge$ Invariant $\rightarrow End_{action}$	14. et P.C.
16. $In_{action} \wedge$ Invariant $\rightarrow$ ($End_{action} \wedge$ **X**Invariant)	12. et 15.
17. $End_{action} \wedge$ **X**Invariant $\rightarrow$ **X**Postcondition	c.
18. $In_{action} \wedge$ Invariant $\rightarrow$ **X**Postcondition	16. et 17.

B2. End_{action} ne se réalise pas

19. $In_{action} \rightarrow$ **X**In_{action}	13., P.C. et hyp B2.
20. $In_{action} \wedge$ Invariant $\rightarrow$ **X**In_{action}	19. et P.C.
21. $In_{action} \wedge$ Invariant $\rightarrow$ **X**($In_{action} \wedge$ Invariant)	12., 20. et P.C.

Nous pouvons reprendre la démonstration au point 12. Par récurrence sur les instants, nous obtiendrons que si l'action se termine, l'invariant sera vrai à l'instant suivant et donc, nous aurons la postcondition.

Terminaison

Pour prouver la terminaison, une légère modification de la méthode de preuve est nécessaire. Nous définissons une fonction H (pouvant dépendre de tous les attributs de l'objet) bornée supérieurement (inférieurement); ensuite, plutôt que de démontrer que la fonction H croît (décroît) strictement, nous démontrons qu'elle ne décroît (croît) pas et que, tant qu'elle est inférieure (supérieure) à la borne, tôt ou tard, elle croîtra (décroîtra) strictement.

Dans le cas où H est bornée supérieurement, nous avons :

a. $In_{action} \rightarrow \mathbf{X}H \geq H$;
b. $In_{action} \wedge H < Borne \rightarrow \mathbf{F}(\mathbf{X}H > H)$;
c. $In_{action} \wedge H \geq Borne \rightarrow \mathbf{F}End_{action}$;

par contre si H est bornée inférieurement, nous aurrons :

a. $In_{action} \rightarrow \mathbf{X}H \leq H$;
b. $In_{action} \wedge H > Borne \rightarrow \mathbf{F}(\mathbf{X}H < H)$;
c. $In_{action} \wedge H \leq Borne \rightarrow \mathbf{F}End_{action}$.

Démontrons que si H est bornée supérieurement et les formules a., b. et c. sont respectées, alors une action commencée, se terminera tôt ou tard.

1. $Beg_{action} \rightarrow \neg In_{action} \wedge (XIn_{action} \vee End_{action})$	Axiome de l'interface
2. $Beg_{action} \rightarrow XIn_{action} \vee End_{action}$	1. et P.C.

Si on a End_{action}, la démonstration est terminée. Supposons que ce n'est pas le cas, dès lors $Beg_{action} \rightarrow XIn_{action}$ est vrai et H aura une certaine valeur notée H_0.

Si H_0 est supérieure ou égale à la borne, nous aurons prouvé que, tôt ou tard, H le sera également.
Supposons donc que $H_0 < Borne$.

3. $In_{action} \rightarrow \mathbf{X}In_{action}$ $\mathcal{W}$ End_{action}	Axiome de l'interface
4. $In_{action} \rightarrow \mathbf{X}H \geq H$	a.
5. $In_{action} \rightarrow \mathbf{X}H \geq H \wedge (\mathbf{X}(\mathbf{X}H \geq H)$ $\mathcal{W}$ $End_{action})$	3. et 4.
6. $In_{action} \wedge H < Borne \rightarrow \mathbf{F}(\mathbf{X}H > H)$	b.
7. $In_{action} \wedge H_0 < Borne \rightarrow \mathbf{F}(\mathbf{X}H > H_0)$	5. et 6.

Par récurrence sur la valeur de H, nous déduisons que tôt ou tard, H sera supérieure ou égale à la borne.

Si H est supérieure ou égale à la borne et "In_{action}" est vrai, d'après c., l'action se terminera tôt ou tard.

Application de cette méthode à notre exemple

Utilisons cette méthode pour démontrer la propriété 3.

Avant de passer à la démonstration proprement dite, prouvons deux propriétés qui nous seront utiles.

Lemme1 : Incrémenter $\rightarrow \exists x\ In_{Ajouter}(x)$

1. Incrémenter $\vee$ Décrémenter $\rightarrow \exists x\ (In_{Ajouter}(x) \vee In_{Soustraire}(x))$	C23
2. Incrémenter $\rightarrow \exists x\ (In_{Ajouter}(x) \vee In_{Soustraire}(x))$	1. et P.C.
3. Incrémenter $\rightarrow \neg\exists x\ In_{Soustraire}(x)$	C36
4. Incrémenter $\rightarrow \exists x\ In_{Ajouter}(x)$	2. et 3.

Lemme2 : Décrémenter $\rightarrow \exists x\ In_{Soustraire}(x)$

1. Incrémenter $\vee$ Décrémenter $\rightarrow \exists x\ (In_{Ajouter}(x) \vee In_{Soustraire}(x))$	C23
2. Décrémenter $\rightarrow \exists x\ (In_{Ajouter}(x) \vee In_{Soustraire}(x))$	1. et P.C.
3. Décrémenter $\rightarrow \neg\exists x\ In_{Ajouter}(x)$	C37
4. Décrémenter $\rightarrow \exists x\ In_{Soustraire}(x)$	2. et 3.

1.A. correction partielle.

Il nous faut prouver que $End_{Ajouter}(x) \rightarrow X(Quantité = Quantité_0 + x)$.
Trouvons un invariant tel que :

$A.1.\ Beg_{Ajouter}(x) \wedge Quantité = Quantité_0 \rightarrow Xinvariant$
$A.2.\ In_{Ajouter}(x) \wedge invariant \rightarrow Xinvariant$
$A.3.\ End_{Ajouter}(x) \wedge Xinvariant \rightarrow X(Quantité = Quantité_0 + x)$

Cet invariant est : $x - Reste = Quantité - Quantité_0$

1.B. Terminaison

$B1.\ In_{Ajouter}(x) \rightarrow XReste \leq Reste$
$B2.\ In_{Ajouter}(x) \wedge Reste > 0 \rightarrow F(XReste < Reste)$
$B3.\ In_{Ajouter}(x) \wedge Reste = 0 \rightarrow FEnd_{Ajouter}(x)$

Dans notre cas, le reste ne peut pas descendre en-dessous de zéro. Démontrons donc que si "$In_{Ajouter}(x)$" est vrai et que le reste est égal à zéro, tôt ou tard, l'action "Ajouter" se terminera.

Détails de la démonstration

1.A. Correction partielle.

1.A.1. Démontrons que si "$Beg_{Ajouter}(x)$" s'exécute, l'invariant "*XReste* = *x* et *XQuantité* = *Quantité*" est vérifié.

1. $Beg_{Ajouter}(x) \rightarrow$ Initialiser(x)	C24
2. Initialiser(x) $\rightarrow$ **X**Reste = x	C6
3. $Beg_{Ajouter}(x) \rightarrow$ XReste = x	**1., 2. et P.C.**
4. Incrémenter $\rightarrow$ $In_{Ajouter}(x)$	Lemme1
5. $In_{Ajouter}(x) \rightarrow \neg$Incrémenter	4. et P.C.
6. $Beg_{Ajouter}(x) \rightarrow \neg\exists y\ In_{Ajouter}(y) \wedge (\mathbf{X}In_{Ajouter}(x) \vee End_{Ajouter}(x))$	C13
7. $Beg_{Ajouter}(x) \rightarrow \neg$Incrémenter	5.,6. et P.C.
8. Décrémenter $\rightarrow \exists x\ In_{Soustraire}(x)$	Lemme2
9. $\neg\exists x\ In_{Soustraire}(x) \rightarrow \neg$Décrémenter	8. et P.C.
10. $Beg_{Ajouter}(x) \rightarrow \neg\exists y\ In_{Soustraire}(y)$	C35
11. $Beg_{Ajouter}(x) \rightarrow \neg$Décrémenter	9., 10. et P.C.
12. $\neg$(Incrémenter $\vee$ Décrémenter) $\rightarrow$ XQuantité = Quantité	C5
13. $Beg_{Ajouter}(x) \rightarrow$ XQuantité = Quantité	**7., 11., 12. et P.C.**
14. $Beg_{Ajouter}(x) \wedge$ (Quantité = $Quantité_0$) $\rightarrow$ X(x - Reste = Quantité - $Quantité_0$)	**3. et 13.**

1.A.2. Pour prouver $In_{Ajouter} \wedge (x - Reste = Quantité - Quantité_0) \rightarrow X(x - Reste = Quantité - Quantité_0)$,
nous montrons que dans tous les cas, si l'invariant est vrai à un instant i, alors, il l'est toujours vrai à l'instant i+1.

Premièrement, observons ce qui se passe lorsque l'action "Incrémenter" s'exécute à l'instant i.

15. Incrémenter $\rightarrow$ Décroître	C27
16. Décroître $\rightarrow$ **X**Reste = prec(Reste)	C7
17. Incrémenter $\rightarrow$ XReste = prec(Reste)	15.,16. et P.C.
18. Incrémenter $\rightarrow$ XQuantité = succ(Quantité)	C1
19. $In_{Ajouter}(x) \wedge$ Incrémenter $\wedge$ (x - Reste = Quantité - $Quantité_0$) $\rightarrow$ X(x - Reste = Quantité - $Quantité_0$)	**17. et 18.**

Ensuite, voyons ce qui se passe lorsque l'action "Incrémenter" ne s'exécute pas à l'instant i.

20. Décrémenter $\rightarrow \neg In_{Ajouter}(x)$	C36
21. $In_{Ajouter}(x) \rightarrow \neg$Décrémenter	20 et P.C.
22. Initialiser(x) $\rightarrow Beg_{Ajouter}(x) \vee Beg_{Soustraire}(x)$	C26
23. $Beg_{Soustraire}(x) \rightarrow \neg In_{Ajouter}(y)$	C34

24. $In_{Ajouter}(y) \rightarrow \neg Beg_{Soustraire}(x)$	23. et P.C.
25. $In_{Ajouter}(y) \rightarrow \neg Beg_{Ajouter}(x)$	6. et P.C.
26. $In_{Ajouter}(x) \rightarrow \neg$Initialiser(y)	22. , 24. et 25.
27. Décroître $\rightarrow$ Incrémenter $\vee$ Décrémenter	C29
28. $In_{Ajouter}(x) \wedge$ Décroître $\rightarrow$ Incrémenter	21. et 27.
29. $In_{Ajouter}(x) \wedge \neg$Incrémenter $\rightarrow \neg$Décroître	28. et P.C.
30. $\neg$(Décroître $\vee$ Initialiser(x)) $\rightarrow$ **X**Reste = Reste	C10
31. $\neg$(Incrémenter $\vee$ Décrémenter) $\rightarrow$ **X**Quantité = Quantité	C5
32. $In_{Ajouter}(x) \wedge \neg$Incrémenter$\wedge$ (x - Reste = Quantité - $Quantité_0$) $\rightarrow$ X(x - Reste = Quantité - $Quantité_0$)	**21.,26., 29., 30 et 31.**

D'après 19. et 32. nous pouvons déduire que lorsque nous sommes dans $In_{Ajouter}$, si l'invariant est vérifié à l'instant i alors il le sera aussi à l'instant i + 1

33. $In_{Ajouter}(x) \wedge$ (x - Reste = Quantité - $Quantité_0$) $\rightarrow$ X(x - Reste = Quantité - $Quantité_0$)	**19., 32. et P.C.**

1.A.3. Pour démontrer que $End_{Ajouter}(x) \wedge X(x - Reste = Quantité - Quantité_0) \rightarrow X(Quantité = Quantité_0 + x)$, il suffit de démontrer que $End_{Ajouter}(x) \rightarrow XReste = 0$ puisque dans ce cas, nous avons $(x - 0 = Quantité - Quantité_0)$ et donc $(Quantité = Quantité_0 + x)$

34. $End_{Ajouter}(x) \rightarrow$ (Reste = 0 $\wedge$ $In_{Ajouter}(x)$)	C32
35. Décroître $\rightarrow$ (Reste > 0) = true	C8
36. Reste = 0 $\rightarrow \neg$Décroître	35. et P.C.
37. $In_{Ajouter}(x) \rightarrow \neg$Initialiser(y)	26
38. $In_{Ajouter}(x) \wedge$ (Reste = 0) $\rightarrow$ **X**Reste = 0	36., 37. et 30.
39. $End_{Ajouter}(x) \rightarrow$ XReste = 0	**34. et 38.**

1.B. Terminaison

1.B.1.	40. Incrémenter $\rightarrow$ **X**Reste = prec(Reste)	17.
	41. $In_{Ajouter}(y) \rightarrow \neg$Initialiser(x)	26.
	42. $In_{Ajouter}(x) \wedge \neg$Incrémenter $\rightarrow \neg$Décroître	29.
	43. $In_{Ajouter}(x) \wedge \neg$Incrémenter $\rightarrow$ **X**Reste = Reste	41. 42. et 30.
	44. $In_{Ajouter}(x) \rightarrow$ XReste $\leq$ Reste	**40. et 43.**
1.B.2.	45. Reste > 0 $\rightarrow$ **F**Décroître	C9
	46. Décroître $\rightarrow$ **X**Reste = prec(Reste)	C7
	47. $\exists x\ In_{Ajouter}(x) \wedge$ Reste > 0 $\rightarrow$ F (XReste < Reste)	**45. et 46.**
1.B.3.	**48. $In_{Ajouter}(x) \wedge$ (Reste = 0) $\rightarrow$ F$End_{Ajouter}(x)$**	**C30**

2.

Dans l'interface de "Ajouter", nous retrouvons les axiomes suivants :

$Beg_{Ajouter}(x) \rightarrow \neg\exists y\ In_{Ajouter}(y) \wedge (\mathbf{X}In_{Ajouter}(x) \vee End_{Ajouter}(x))$

$In_{Ajouter}(x) \rightarrow ((\mathbf{X}In_{Ajouter}(x))\ \mathcal{W}\ End_{Ajouter}(x))$

De cela, nous déduisons que :

$Beg_{Ajouter}(x) \rightarrow \mathbf{X}In_{Ajouter}(x))\ \mathcal{W}\ End_{Ajouter}(x)$

A fortiori, nous avons :

$Beg_{Ajouter}(x) \wedge Quantité = Quantité_0 \rightarrow \mathbf{X}In_{Ajouter}(x))\ \mathcal{W}\ End_{Ajouter}(x)$

Nous avons démontrer au poinrt A. que si on commençait l'action "Ajouter" avec Quantité = $Quantité_0$, tôt ou tard, celle-ci se terminera et nous aurons $\mathbf{X}$(Quantité = $Quantité_0$ + x)). Nous avons donc démontré que :

$\mathbf{Beg_{Ajouter}(x) \wedge Quantité = Quantité_0 \rightarrow XIn_{Ajouter}(x))\ \mathcal{W}\ (End_{Ajouter}(x) \wedge X(Quantité = Quantité_0 + x))}$

4.5.2. Seconde approche

Dans la première approche, nous avons utilisé la méthode "classique" de connection des objets. En examinant les axiomes utilisés pour décrire la boucle, il apparait que si $In_{Ajouter}$ ($In_{Soustraire}$) est vrai alors les actions "Incrémenter" ("Décrémenter") et "Décroître" sont synchronisées. De cette constatation nous est venue l'idée de synchroniser ces actions non plus grâce à des axiomes mais grâce à la technique des Canaux expliquée au chapitre 3.

Faisons cela en trois étapes :

1°) dupliquons l'objet "R": l'un ("$R_{Ajouter}$") correspondra à l'action abstraite "Ajouter", et l'autre ("$R_{Soustraire}$") à l'action "Soustraire";

2°) synchronisons les actions "Initialiser" et "Décroître" de "$R_{Ajouter}$" avec les actions "$Beg_{Ajouter}$" et "Incrémenter" ;

3°) synchronisons les actions "Initialiser" et "Décroître" de "$R_{Soustraire}$" avec les actions "$Beg_{Soustraire}$" et "Décrémenter".

4.5.2.1. Construction du corps

La structure de conception selon cette deuxième approche est reprise à la figure 4.2.

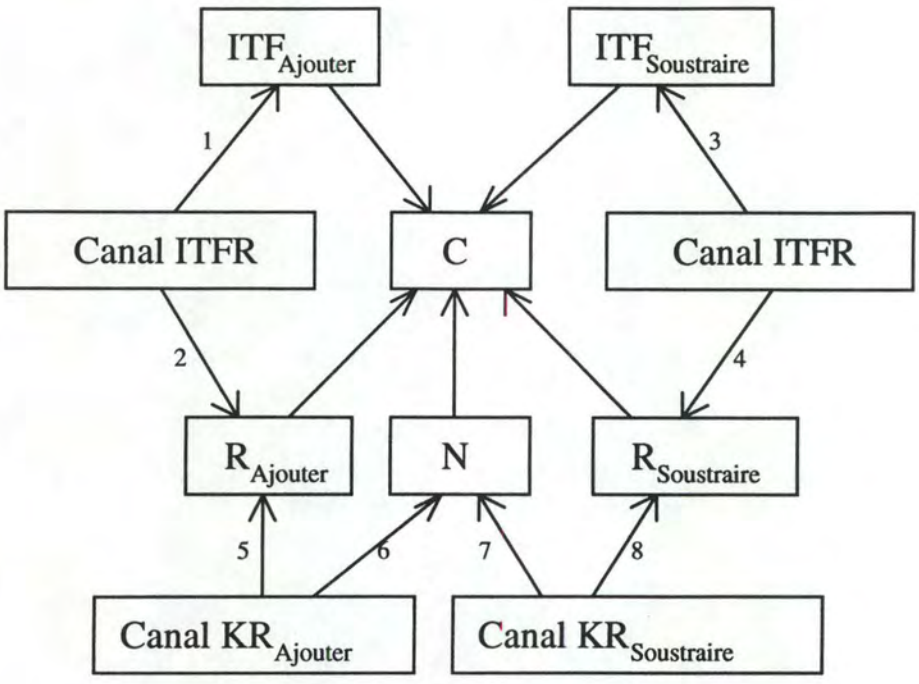

Fig. 4.2. *: Structure de conception pour "N" selon la seconde approche.*

La description des composants sera :

Canal ITFR
Sorts : BOOL, NAT
Symboles d'opération : /
Symboles d'attribut : /
Symboles d'action : Init(NAT)
Axiomes : /

Canal $NR_{Ajouter}$

Sorts : BOOL

Symboles d'opération : /

Symboles d'attribut : /

Actions : Incr

Axiomes : /

Canal $NR_{Soustraire}$

Sorts : BOOL

Symboles d'opération : /

Symboles d'attribut : /

Symboles d'action : Decr

Axiomes : /

Pour synchroniser les actions selon la technique des Canaux, nous devons définir les morphismes reliant les différents objets.

1

Relation: Canal ITF R $\rightarrow$ $ITF_{Ajouter}$
Sorts : ID_{BOOL}, ID_{NAT}
Symboles d'action : Init(NAT)$\rightarrow$ $Beg_{Ajouter}$(NAT)

2

Relation : Canal ITF R → $R_{Ajouter}$
Sorts : ID_{BOOL}, ID_{NAT}
Symboles d'action : Init(NAT) → Initialiser(NAT)

3

Relation : Canal ITF R → $ITF_{Soustraire}$
Sorts : ID_{BOOL}, ID_{NAT}
Symboles d'action : Init(NAT) → $Beg_{Soustraire}$(NAT)

4

Relation: Canal ITF R → $R_{Soustraire}$
Sorts : ID_{BOOL}, ID_{NAT}
Symboles d'action : Init(NAT) → Initialiser(NAT)

5

Relation : Canal $NR_{Ajouter}$ → N
Sorts : ID_{BOOL}
Symboles d'action : Incr → Incrémenter

6

Relation : Canal $NR_{Ajouter}$ → $R_{Ajouter}$
Sorts : ID_{BOOL}
Symboles d'action : Incr → Décroître

7

Relation : $CanalNR_{Soustraire} \rightarrow N$
Sorts : ID_{BOOL}
Symboles d'action : Decr → Décrémenter

8

Relation : $CanalNR_{Soustraire} \rightarrow R_{Soustraire}$
Sorts : ID_{BOOL}
Symboles d'action : Decr → Décroître

Par cette approche, la construction du corps est un peu différente. On n'a plus besoin d'utiliser des axiomes pour synchroniser les actions entre elles puisque d'après la théorie des Canaux, les actions qui sont synchronisées sont représentées par le même symbole d'action. Comme nous avons introduit deux objets "R", nous aurons dans le corps deux attributs "Reste1" et "Reste2".

C

Sorts : NAT, BOOL

Symboles d'opération : Fonctions et constantes habituelles pour les booléens.
prec, succ : NAT → NAT
>, ≤ : NAT × NAT → BOOL
0 : → NAT

Symboles d'attribut :
Quantité : NAT
Reste1 : NAT
Reste2 : NAT
$In_{Ajouter}$ (NAT): BOOL
$In_{Soustraire}$ (NAT): BOOL

Symboles d'action : Incrémenter
Décrémenter
$Beg_{Ajouter}(NAT)$
$Beg_{Soustraire}(NAT)$
$End_{Ajouter}(NAT)$
$End_{Soustraire}(NAT)$

Axiomes :

{ Axiomes de "N". }

C1) Incrémenter $\rightarrow$ **X**Quantité = succ(Quantité)
C2) Décrémenter $\rightarrow$ **X**Quantité = prec(Quantité)
C3) Décrémenter $\rightarrow$ Quantité > 0
C4) **BEG** $\rightarrow$ Quantité = 0
C5) $\neg$(Incrémenter $\vee$ Décrémenter) $\rightarrow$ **X**Quantité = Quantité

{ Axiomes de "$R_{Ajouter}$". }

C6) $Beg_{Ajouter}(x) \rightarrow$ **X**Reste1 = x
C7) Incrémenter $\rightarrow$ **X**Reste1 = prec(Reste1)
C8) Incrémenter $\rightarrow$ (Reste1 > 0) = true
C9) Reste1 > 0 $\rightarrow$ **F**Incrémenter
C10) $\neg$(Incrémenter $\vee$ $Beg_{Ajouter}(x)$) $\rightarrow$ **X**Reste1 = Reste1

{ Axiomes de "$R_{Soustraire}$". }

C11) $Beg_{Soustraire}(x) \rightarrow$ XReste2 = x
C12) Décrémenter $\rightarrow$ **X**Reste2 = prec(Reste2)
C13) Décrémenter $\rightarrow$ (Reste2 > 0) = true
C14) Reste2 > 0 $\rightarrow$ **F**Incrémenter
C15) $\neg$(Décrémenter $\vee$ $Beg_{Soustraire}(x)$) $\rightarrow$ **X**Reste2 = Reste2

{ Axiomes de "$ITF_{Ajouter}$". }

C16) **BEG** $\rightarrow \neg In_{Ajouter}$
C17) $\neg In_{Ajouter}(x) \rightarrow \exists x\ ((\mathbf{X}\neg In_{Ajouter}(x))\ \mathcal{W}\ Beg_{Ajouter}(x))$
C18) $Beg_{Ajouter}(x) \rightarrow \neg In_{Ajouter}(x) \wedge (\mathbf{X}In_{Ajouter}(x) \vee End_{Ajouter}(x))$
C19) $In_{Ajouter}(x) \rightarrow \exists x\ ((\mathbf{X}In_{Ajouter}(x))\ \mathcal{W}\ End_{Ajouter}(x))$
C20) $End_{Ajouter}(x) \rightarrow (In_{Ajouter}(x) \vee Beg_{Ajouter}(x)) \wedge \mathbf{X}\neg In_{Ajouter}(x)$
C21) $\neg\exists x\ (Beg_{Ajouter}(x) \vee End_{Ajouter}(x)) \rightarrow \mathbf{X}In_{Ajouter}(x) = In_{Ajouter}(x)$

{ Axiomes de "$ITF_{Soustraire}$". }

C22) **BEG** $\rightarrow \neg In_{Soustraire}(x)$
C23) $\neg In_{Soustraire}(x) \rightarrow \exists x\ ((\mathbf{X}\neg In_{Soustraire}(x))\ \mathcal{W}\ Beg_{Soustraire}(x))$
C24) $Beg_{Soustraire}(x) \rightarrow \neg In_{Soustraire}(x) \wedge (\mathbf{X}In_{Soustraire}(x) \vee End_{Soustraire}(x))$
C25) $In_{Soustraire}(x) \rightarrow \exists x\ ((\mathbf{X}In_{Soustraire}(x))\ \mathcal{W}\ End_{Soustraire}(x))$
C26) $End_{Soustraire}(x) \rightarrow (In_{Soustraire}(x) \vee Beg_{Soustraire}(x)) \wedge \mathbf{X}\neg In_{Soustraire}(x)$
C27) $\neg\exists x(Beg_{Soustraire}(x) \vee End_{Soustraire}(x)) \rightarrow \mathbf{X}In_{Soustraire}(x) = In_{Soustraire}(x)$

{ Les axiomes suivants implémentent la condition de localité de A. }

C28) Incrémenter $\vee$ Décrémenter $\rightarrow$ ($In_{Ajouter}(x) \vee In_{Soustraire}(x)$)

{ Ces axiomes assurent le fait que les actions "Ajouter" et "Soustraire" ne peuvent s'exécuter en même temps. }

C29) $Beg_{Ajouter}(x) \rightarrow \neg(in_{Soustraire}(y) \vee Beg_{Soustraire}(y))$
C30) $Beg_{Soustraire}(x) \rightarrow \neg(in_{Ajouter}(y) \vee Beg_{Ajouter}(y))$
C31) Incrémenter $\rightarrow \neg(in_{Soustraire}(x) \vee Beg_{Soustraire}(x))$
C32) Décrémenter $\rightarrow \neg(in_{Ajouter}(x) \vee Beg_{Ajouter}(x))$

{ Les axiomes suivants indiquent que si on est en train d'exécuter l'action "Ajouter" ("Soustraire") et si l'attribut "Reste1" ("Reste2") est égal à zéro, tôt ou tard, on terminera l'action "Ajouter" ("Soustraire") mais celle-ci ne se terminera qu'à cette condition. }

C33) $End_{Ajouter}(x) \rightarrow (Reste1 = 0 \wedge in_{Ajouter}(x))$
C34) $End_{Soustraire}(x) \rightarrow (Reste2 = 0 \wedge in_{Soustraire}(x))$
C35) $in_{Ajouter}(x) \wedge (Reste1 = 0) \rightarrow FEnd_{Ajouter}(x)$
C36) $in_{Soustraire}(x) \wedge (Reste2 = 0) \rightarrow FEnd_{Soustraire}(x)$

{ Axiome de sécurité. }

C37) $Beg_{Soustraire}(x) \rightarrow x \leq$ Quantité

Comme lors de la première approche nous allons tenter de dégager un schéma général de construction du corps lorsque les morphismes sont utilisés pour réaliser les synchronisations.

Pour construire le corps, il suffit, encore une fois, de faire la somme des objets constituants. Nous avons introduit, pour chaque action réclamant l'utilisation d'une boucle, un nouvel objet "Reste". Une des différences avec la première approche est que nous aurons dans le corps les axiomes de chacun de ces objets. En ce qui concerne les familles que nous avions définies dans la première approche, nous les retrouvons toutes à l'exception des sous-familles 3)a) et 3)b) (initialisation et corps de la boucle). Ce n'est pas étonnant puisque ces axiomes avaient pour but de synchroniser des actions, or ici les synchronismes sont réalisés via les morphismes .

4.5.2.2. Démonstration des propriétés

L'objet construit doit respecter les propriétés de l'objet A. Nous devons donc démontrer que :

1. ***BEG*** $\rightarrow$ *Quantité = 0*
2. *Soustraire(x)* $\rightarrow$ *(x* $\leq$ *Quantité) = true*
3. *Ajouter(x)* $\rightarrow$ ***XQuantité*** *= Quantité + x*
4. *Soustraire(x)* $\rightarrow$ ***XQuantité*** *= Quantité + x*

La propriété 1. se déduit de l'axiome C4 et de la règle 4•.
La propriété 2. se déduit de l'axiome C37 et de la règle 2•.

Pour les deux dernières propriétés, nous avons le même type de preuve que dans la première approche mais comme nous avons introduit deux attributs "Reste1" et "Reste2", l'invariant devient : x - *Reste1* = *Quantité* - *Quantité*$_0$

Grâce à la synchronisation des actions, certains pas triviaux de la démonstration seront supprimés.

Détails de la démonstration

La démonstration des lemmes 1 et 2 se fait de la même manière que dans la première approche à partir des axiomes C28, C31 et C32. Nous pourrons donc les utiliser dans la démonstration.

A. Correction partielle

A.1. Pour démontrer ce point, nous devons prouver que lorsque l'action "$Beg_{Ajouter}(x)$" s'exécute, nous avons ***XReste1*** = x et ***XQuantité*** = *Quantité* et donc, l'invariant est vérifié.

1. $Beg_{Ajouter}(x) \rightarrow$ XReste1 = x.	**C6**
2. Incrémenter $\rightarrow$ $In_{Ajouter}$	Lemme1
3. $\neg In_{Ajouter} \rightarrow \neg$Incrémenter	2. et P.C.
4. $Beg_{Ajouter}(x) \rightarrow \neg In_{Ajouter} \wedge (\mathbf{X}In_{Ajouter} \vee End_{Ajouter}(x))$	C17
5. $Beg_{Ajouter}(x) \rightarrow \neg$Incrémenter	3., 4. et P.C.
6. Décrémenter $\rightarrow$ $In_{Soustraire}$	Lemme2
7. $\neg In_{Soustraire} \rightarrow \neg$Décrémenter	6. et P.C.
8. $Beg_{Ajouter}(x) \rightarrow \neg In_{Soustraire}$	C29
9. $Beg_{Ajouter}(x) \rightarrow \neg$Décrémenter	7., 8. et P.C.
10. $\neg$(Incrémenter $\vee$ Décrémenter) $\rightarrow$ XQuantité = Quantité	C5
11. $Beg_{Ajouter}(x) \rightarrow$ XQuantité = Quantité	**5., 9. et 10.**
12. $Beg_{Ajouter}(x) \wedge$ (Quantité$_0$ = Quantité) $\rightarrow$ X(x - Reste1 = Quantité - Quantité$_0$)	**1. et 11.**

A.2. Pour prouver $In_{Ajouter} \wedge (x - Reste1 = Quantité - Quantité_0) \rightarrow X(x - Reste1 = Quantité - Quantité_0)$, montrons que dans tous les cas, si l'invariant est vérifié à un instant i, il l'est toujours à l'instant i + 1.

Dans un premier temps, observons ce qui se passe lorsque l'action "Incrémenter" s'exécute.

13. Incrémenter $\rightarrow$ **X**Reste1 = prec(Reste1)	C7
14. Incrémenter $\rightarrow$ **X**Quantité = succ(Quantité)	C1
15. $In_{Ajouter}$ $\wedge$ Incrémenter $\wedge$ (x - Reste1 = Quantité - $Quantité_0$) $\rightarrow$ X(x - Reste1 = Quantité - $Quantité_0$)	**13. et 14.**

Examinons maintenant le cas où l'action "Incrémenter" ne s'exécute pas.

16. Décrémenter $\rightarrow \neg In_{Ajouter}$	C32
17. $In_{Ajouter} \rightarrow \neg$Décrémenter	16. et P.C.
18. $In_{Ajouter} \rightarrow \neg Beg_{Ajouter}(x)$	4. et P.C.
19. $\neg$(Incrémenter $\vee$ $Beg_{Ajouter}(x)$) $\rightarrow$ **X**Reste1 = Reste1	C10
20. $\neg$(Incrémenter $\vee$ Décrémenter) $\rightarrow$ **X**Quantité = Quantité	C5
21. $In_{Ajouter}$ $\wedge$ $\neg$Incrémenter $\wedge$ (x - Reste1 = Quantité - $Quantité_0$) $\rightarrow$ X(x - Reste1 = Quantité - $Quantité_0$)	**17., 18., 19. et 20.**

D'après 15. et 21., si l'action "Ajouter" est en court ($In_{Ajouter}$ est vrai) et que l'invariant est vérifié à l'instant i, alors à l'instant i + 1, l'invariant sera toujours respecté.

22. $In_{Ajouter}$ $\wedge$ (x - Reste1 = Quantité - $Quantité_0$) $\rightarrow$ X(x - Reste1 = Quantité - $Quantité_0$)	**15. et 21.**

A.3. Pour démontrer que $End_{Ajouter}(x) \wedge X(x - Reste1 = Quantité - Quantité_0) \rightarrow X(Quantité = Quantité_0 + x)$, il suffit de démontrer que $End_{Ajouter}(x) \rightarrow XReste1 = 0$ puisque dans ce cas, $(x - 0 = Quantité - Quantité_0)$ est vrai et donc $(Quantité = Quantité_0 + x)$ l'est aussi.

23. $End_{Ajouter} \rightarrow$ (Reste1 = 0 $\wedge$ $In_{Ajouter}$)	C33
24. Incrémenter $\rightarrow$ (Reste1 > 0) = true	C8
25. Reste1 = 0 $\rightarrow \neg$Incrémenter	24. et P.C.
26. Reste1 = 0 $\wedge$ $In_{Ajouter} \rightarrow$ **X**Reste1 = 0	18., 25. et 19.
27. $End_{Ajouter} \rightarrow$ XReste1 = 0	**23. et 26.**

B. Terminaison

B1.	28. Incrémenter $\rightarrow$ **X**Reste1 = prec(Reste1)	C7
	29. $In_{Ajouter} \rightarrow \neg Beg_{Ajouter}(x)$	18.
	30. $In_{Ajouter}$ $\wedge$ $\neg$Incrémenter $\rightarrow$ **X**Reste1 = Reste1	29. et 19.
	31. $In_{Ajouter} \rightarrow$ XReste1 $\leq$ Reste1	**29. et 30.**

B2.	32. Reste1 > 0 → **F**Incrémenter	C9
	33. Incrémenter → **X**Reste1 = prec(Reste1)	C7
	34. $In_{Ajouter}$ ∧ Reste1 > 0 → F (XReste1 < Reste1)	**32., 33. et P.C.**

La point 2. se démontre de la même manière que dans la première approche.

Chapitre 5

Modifications de la théorie

5.1. Introduction

Lors du développement des exemples 3.1 et 4.1, nous avons remarqué que certaines modifications pouvaient être faites afin d'améliorer la théorie. Au point 3.2.2, un quatrième type d'axiomes a été introduit, de même au point 4.5.1.2., la règle de transition de Changement a du être modifiée. Dans ce chapitre, nous allons présenter deux autres modifications apportées à la théorie.

Dans la première section, nous verrons comment nous avons supprimé certains axiomes des interfaces des actions. Ensuite, nous verrons comment nous avons optimisé la gestion de la concurrence du niveau concret et, dans la section suivante, les répercussions sur la théorie. Enfin, dans un cinquième point, nous développerons un exemple qui illustrera les avantages des modifications apportées.

5.2. Modification des axiomes des interfaces

Le chapitre 3 a montré que dans la théorie initiale, l'interface "ITFa" comprend deux actions distinctes beg_a et end_a, et un attribut in_a. Ces éléments satisfont les conditions suivantes :

1. $ITF_a \Rightarrow (\mathbf{BEG} \rightarrow \neg in_a)$
2. $ITF_a \Rightarrow (beg_a \rightarrow \neg in_a \wedge (\mathbf{X} in_a \vee end_a))$

3. $ITF_a \Rightarrow (end_a \rightarrow (in_a \vee beg_a) \wedge \mathbf{X}\neg in_a)$

4. $ITF_a \Rightarrow (in_a \rightarrow ((\mathbf{X}in_a) \mathcal{W} end_a))$

5. $ITF_a \Rightarrow (\neg in_a \rightarrow ((\mathbf{X}\neg in_a) \mathcal{W} beg_a))$

Lorsqu'on analyse de près ces axiomes et qu'on tient compte de la condition de localité ($\neg (beg_a \vee end_a) \rightarrow \mathbf{X}in_a = in_a$), on remarque que les axiomes 4. et 5. sont inutiles. En effet, ils peuvent être déduits de 1., 2., 3. et de la condition de localité.

Preuve

A. $\neg in_a \rightarrow ((\mathbf{X}\neg in_a) \mathcal{W} beg_a)$

Prenons la règle[22] :

$[X \wedge \Box(X \supset (B \vee (A \wedge OX)))] \supset A \mathcal{U} B$

où X est une formule quelconque et "$\mathcal{U}$" est l'opérateur désignant le "untill" faible. Les opérateurs $\Box$ et O signifient "toujours" et "à l'instant suivant".

Dans notre formalisme cela donne :

$[C \wedge \Box(C \rightarrow (B \vee (A \wedge \mathbf{X}C)))] \rightarrow A \mathcal{W} B$

C étant une formule arbitraire.

Pour notre preuve, nous allons prendre C = "$\neg in_a$", A = "$\mathbf{X}\neg in_a$" et B = "beg_a". Nous aurons donc :

$[\neg in_a \wedge \Box(\neg in_a \rightarrow (beg_a \vee (\mathbf{X}\neg in_a \wedge \mathbf{X}\neg in_a)))] \rightarrow \mathbf{X}\neg in_a \mathcal{W} beg_a$

Dans le cas où "$\neg in_a$" est faux, la démonstration de A est immédiate. Supposons que "$\neg in_a$" soit vrai.

$\Rightarrow \Box(\neg in_a \rightarrow (beg_a \vee (\mathbf{X}\neg in_a \wedge \mathbf{X}\neg in_a))) \rightarrow \mathbf{X}\neg in_a \mathcal{W} beg_a$ (comme "$\neg in_a$" est vrai, nous pouvons le supprimer dans une conjonction)

$\Rightarrow \Box(\neg in_a \rightarrow (beg_a \vee \mathbf{X}\neg in_a)) \rightarrow \mathbf{X}\neg in_a \mathcal{W} beg_a$ ($B \wedge B \equiv B$)

Il nous reste à démontrer la formule "$\neg in_a \rightarrow \mathbf{X}\neg in_a \vee beg_a$".

Quatre cas peuvent se présenter : aucune action ne s'exécute, seule l'action "end_a" s'exécute, seule l'action "beg_a" s'exécute ou les deux actions s'exécutent.

---

[22] Tirée de [Thayse 89, p184].

- Si le premier cas se produit, nous avons :

1. $\neg(beg_a \vee end_a) \rightarrow Xin_a = in_a$	Localité
2. $\neg in_a \rightarrow \mathbf{X}\neg in_a$	1.
3. $\neg in_a \rightarrow \mathbf{X}\neg in_a \vee beg_a$	2. et P.C.

- Le deuxième cas ne peut se produire car nous avons :

1. $end_a \rightarrow (in_a \vee beg_a) \wedge \mathbf{X}\neg in_a$	Axiome 3.
2. $\neg(in_a \vee beg_a) \rightarrow \neg end_a$	1. et P.C.

- Si un des deux derniers cas se produit, l'action "beg_a" s'exécute et donc,

"$\neg in_a \rightarrow \ \mathbf{X}\neg in_a \vee \ beg_a$" est vérifié.

B. $in_a \rightarrow ((\mathbf{X}in_a) \ \mathcal{W} \ end_a$

La preuve est identique à celle du point A. Les quatre mêmes cas peuvent se produire.

- Si le premier cas se produit, nous avons :

1. $\neg(beg_a \vee end_a) \rightarrow Xin_a = in_a$	Localité
2. $in_a \rightarrow \mathbf{X}in_a$	1.
3. $in_a \rightarrow \mathbf{X}in_a \vee end_a$	2. et P.C.

- Le troisième cas ne peut se produire car nous avons :

1. $beg_a \rightarrow \neg in_a \wedge (\mathbf{X}in_a \vee end_a)$	Axiome 2.
2. $in_a \rightarrow \neg beg_a$	1. et P.C.

- Si le deuxième ou le dernier cas se produisent, l'action "end_a" s'exécute et donc,

"$in_a \rightarrow \ \mathbf{X}in_a \vee \ end_a$" est vérifié.

5.3. Optimisation de la gestion de la concurrence et répercussions sur la théorie

Dans le chapitre 3, section 3.5.3, nous avons décrit les conditions qui sont imposées pour respecter l'intégrité de l'implémentation des actions et des instants abstraits. Elles sont très générales mais peuvent être affinées. En effet, dans l'exemple choisi les conditions imposaient qu'aucun attribut ne puisse être changé dans l'intervalle [beg(a),beg(b)]. Or il se peut que certains attributs ne soient ni lus ni modifiés lors de l'intervalle [beg(b),end(b)], donc qu'il ne soient pas utilisés par l'action "b". Dans ce cas, l'exécution de l'action "a" pourrait commencer plus tôt, certaines actions pouvant déjà être exécutées. Il serait donc intéressant d'avoir pour chaque action l'ensemble des attributs qu'elle lit et ceux qu'elle modifie.

5.3.1. Extension de la signature d'objet

Nous allons voir comment intégrer ce nouveau concept au sein des objets et observer les changements éventuels.

La signature d'objet, présentée au chapitre 3, est un triplet qui comprend l'univers, les symboles d'attribut et les symboles d'action. Nous ajoutons deux nouveaux ensembles Θ et Ξ dont le but sera de définir les relations de lecture et d'écriture entre les attributs et les actions. La définition de la signature d'objet comprendra ces deux nouveaux ensembles.

Définition 5.3.1.1 (Signature d'objet)

La signature d'objet sera un quintuplet $(\Sigma, A, \Gamma, \Theta, \Xi)$ où

- $\Sigma = (S,\Omega)$ est une signature (l'*univers*) dans le sens algébrique habituel[23].
- A est une famille indexée par $S^* \times S$ de symboles d'attribut.
- Γ est une famille indexée par S^* de symboles d'action.
- Θ est une famille indexée par A de symboles d'action.
- Ξ est une famille indexée par A de symboles d'action.

Les familles indexées Ω, A et Γ portent sur des ensembles finis et disjoints.

Propriétés de Θ et Ξ :

1. $\forall$ att $\in A^\bullet$, $\Theta(att) \subseteq \Gamma$.
2. $\forall$ att $\in A^\bullet$, $\Xi(att) \subseteq \Gamma$.

5.3.2. Raffinement de la condition de localité

A. Définition de la localité

La condition de localité vue supra spécifiait que si aucune action de l'objet ne s'exécutait, les attributs de cet objet restaient inchangés.

Nous raffinons maintenant cette condition en donnant une localité propre à chaque attribut. Pour tout attribut "att", si aucune action de Ξ(att) ne se produit, alors, "att" demeure inchangé.

Cela s'exprime par la formule suivante :

$$\forall \text{ att} \in A^\bullet \Big(\bigvee_{g \in \Xi_{(att)}} (\exists x_g) g(x_g) \Big) \vee \big(\forall x_{att} (X att(x_{att}) = att(x_{att})) \big)$$

où pour tout symbole u, x_u est un vecteur de variables des sorts appropriés.

---

[23] Repris de [Ehrig & Marh 85].

B. Nouvelle structure d'interprétation

Au chapitre 3, nous avons vu qu'une structure d'interpretation pour un objet est donnée par un algèbre qui interprète les paramètres de l'univers, une projection qui donne les valeurs prises par les attributs à chaque instant, et une projection qui donne les actions qui s'exécutent à chaque instant.

Avec l'apparition des ensembles Ξ et Θ, la structure d'interprétation initiale possède une propriété supplémentaire. Rappelons la définition :

Définition 5.3.2.1. (θ-structures d'interprétation)
Une θ-structure d'interprétation I pour une signature $\theta = (\Sigma, A, \Gamma, \Xi, \Theta)$ est un quintuplet (U, A, G) où :

- U est un Σ-algebre, c'est à dire qu'il assigne à tout symbole $s \in S$ un ensemble s_U, et à chaque symbole de fonction $f \in \Omega_{<s_1,...,s_n>,s}$ une fonction $f_U : s_{1U} \times ... \times s_{nU} \rightarrow s_U$.
- A projette $f \in A_{<s_1,...,s_n>,s}$ vers $A(f) : s_{1U} \times ... \times s_{nU} \times N_0 \rightarrow s_U$. Nous utiliserons la notation $A(f)(i)$ pour $i \in N_0$ pour dénoter la fonction $A(f)(b_1,...,b_n,i)$.
- G projette $g \in \Gamma_{<s_1,...,s_n>}$ vers $G(g) : s_{1U} \times ... \times s_{nU} \rightarrow P(N_0)$.

Où N_0 représente l'ensemble des naturels sans le zéro. Prenons les nombres naturels pour représenter le domaine temporel, c'est à dire un temps linéaire et discret. La projection A donne la valeur des attributs à chaque instant. On dénote une action par l'ensemble des instants durant lesquels l'action s'exécute.

La structure d'interprétation doit vérifier la propriété suivante :
3. $\forall$ att $\in A^{\bullet}$ et $\forall$ act $\in \Gamma^{\bullet}$, si act $\notin \Xi_{att}$, act $\wedge \neg(\bigvee_{g \in \Xi(att)} g) \rightarrow \mathbf{X}att = att$.

C. Equivalence avec la définition initiale

Notre définition de la localité est juste un raffinement de celle présentée par J.L. Fiadeiro et T. Maibaum. Elle nous facilite les preuves mais reste équivalente à l'ancienne définition. C'est ce que nous démontrons dans ce point.

Supposons un objet où :

⇨ $A^\bullet = \{att_1,...,att_n\}$
⇨ $\Gamma^\bullet = \{act_1,...,act_m\}$
- $\forall\ att_i \in A^\bullet$, $\Xi(att_i) = \{act_{i_1},...,act_{i_p}\}$
avec act_{i_j} est la $j^{ème}$ action qui peut modifier l'attribut att_i. On a :
$1 \leq i \leq n$, $1 \leq p \leq m$, et $1 \leq i_1 < ... < i_p \leq m$

Montrons maintenant l'équivalence des deux définitions de la condition de localité pour cet objet.

$\neg (act_1 \vee ... \vee act_m) \rightarrow \mathbf{X}att_1 = att_1 \wedge ... \wedge \mathbf{X}att_n = att_n$
(Ancienne définition de la localité pour notre objet) $\Leftrightarrow$
$\neg (act_{1_p} \vee ... \vee act_{1_p}) \rightarrow \mathbf{X}att_1 = att_1$
...
$\neg (act_{n_1} \vee ... \vee act_{n_p}) \rightarrow \mathbf{X}att_n = att_n$
(Nouvelle définition de la localité pour le même objet)

Démonstration :

$\Rightarrow$

A partir des règles du calcul propositionnel, nous pouvons scinder la formule de manière à obtenir :

$\neg (act_1 \vee ... \vee act_m) \rightarrow \mathbf{X}att_1 = att_1$
...
$\neg (act_1 \vee ... \vee act_m) \rightarrow \mathbf{X}att_n = att_n$

D'après la propriété 3. qui porte sur Ξ, les actions qui n'appartiennent pas à Ξ_{att_i} peuvent s'exécuter. Etant donné que leur exécution ou non ne modifie pas la valeur de vérité de la formule, on peut les supprimer. Nous obtenons :

$\neg (act_{1_1} \vee ... \vee act_{p_1}) \wedge \ \rightarrow \mathbf{X}att_1 = att_1$
...
$\neg (act_{1_n} \vee ... \vee act_{p_n}) \rightarrow \mathbf{X}att_n = att_n$

et donc,

$\neg (act_1 \vee ... \vee act_m) \rightarrow \mathbf{X}att_1 = att_1 \wedge ... \wedge \mathbf{X}att_n = att_n$
$\Rightarrow \neg (act_{1_1} \vee ... \vee act_{p_1}) \rightarrow \mathbf{X}att_1 = att_1$
...
$\neg (act_{1_n} \vee ... \vee act_{p_n}) \rightarrow \mathbf{X}att_n = att_n$

$\Leftarrow$

Par hypothèse

$$\neg (act_{1_1} \vee ... \vee act_{p_1}) \rightarrow \mathbf{X}att_1 = att_1$$
$$...$$
$$\neg (act_{1_n} \vee ... \vee act_{p_n}) \rightarrow \mathbf{X}att_n = att_n$$

et a fortiori, les formules suivantes sont vraies :

$$\neg (act_1 \vee ... \vee act_m) \rightarrow \mathbf{X}att_1 = att_1$$
$$...$$
$$\neg (act_1 \vee ... \vee act_m) \rightarrow \mathbf{X}att_n = att_n$$

D'où

$$\neg (act_1 \vee ... \vee act_m) \rightarrow \mathbf{X}att_1 = att_1 \wedge ... \wedge \mathbf{X}att_n = att_n$$

D. Nouvelle définition du morphisme

La nouvelle définition de la condition de localité et l'apparition des ensembles Θ et Ξ entraîne une modification de la définition de morphisme.

Définition 5.3.2.2 (Morphisme de signature) :

Etant donné deux signatures d'objets $\theta_1 = (\Sigma_1, A_1, \Gamma_1, \Theta_1, \Xi_1)$ et $\theta_2 = (\Sigma_2, A_2, \Gamma_2, \Theta_2, \Xi_2)$, un morphisme σ de θ_1 vers θ_2 consiste en :

- Un morphisme de signatures algébriques $\sigma_v : \Sigma_1 \rightarrow \Sigma_2$
- Pour chaque $f : s_1,...,s_n \rightarrow s$ dans A_1 un symbole d'attribut $\sigma_\alpha(f) : \sigma_v(s_1),..s_1\sigma_v(s_n) \rightarrow \sigma_v(s)$ dans A_2.
- Pour chaque $g : s_1,...,s_n$ dans Γ_1 un symbole d'action $\sigma_\gamma(g) : \sigma_v(s_1),..,\sigma_v(s_n)$ dans Γ_2.
- Pour chaque $att \in A_1^\bullet$, si $\Theta_1(att) = \{g_1,...,g_n\}$, on aura un élément $\sigma_\alpha(att)$ tel que $\Theta_2(\sigma\alpha(att)) \subseteq \{\sigma_\gamma(g_1),...,\sigma_\gamma(g_n)\}$.
- Pour chaque $att \in A_1^\bullet$, si $\Xi_1(att) = (g_1,...,g_n)$, on aura un élément $\sigma_\alpha(att)$ tel que $\Xi_2(\sigma\alpha(att)) \subseteq \{\sigma_\gamma(g_1),...,\sigma_\gamma(g_n)\}$.

Avec la nouvelle définition du morphisme de signature, la condition de localité de chacun des objets sources est automatiquement comprise dans la condition de localité de l'objet cible. Il sera donc inutile de donner leur traduction de manière explicite dans les axiomes.

Définition 5.3.2.3. (Morphisme de descriptions) :

Etant donné deux descriptions d'objets (θ_1, Φ_1) et (θ_2, Φ_2), un morphisme σ : $(\theta_1, \Phi_1) \rightarrow (\theta_2, \Phi_2)$ est un morphisme de signatures $\sigma : \theta_1 \rightarrow \theta_2$ tel que :

- $(\Phi_2 \Rightarrow_{\theta 2} \sigma (p))$ est valide pour tout $p \in \Phi_1$

E. Exemple 5.1.

Afin d'illustrer les avantages du raffinement de la condition de localité, prenons l'objet "SYSTEME" de l'exemple 3.1 et démontrons la propriété "*Cderéalisée = Nbrcde - Longeurliste*".

Les modifications apportées entraînent une spécifications des objets "COMMANDE" et "STOCK" un peu différente.

COMMANDE

Sorts : NAT, QUEUE, BOOL

Symboles d'opération : filevide : QUEUE → BOOL
in : QUEUE × NAT → QUEUE
out : QUEUE → QUEUE
first : QUEUE → NAT
+, - : NAT × NAT → NAT

Symboles d'attribut : File : QUEUE
Qcommandée : NAT
Nbrereçue : NAT
Longueurfile : NAT

Symboles d'action : Ajouter(NAT)
Retirer(NAT)

Attributs lus : File : Ajouter, Retirer
Qcommandée : /
Nbrereçue : Ajouter
Longueurfile : Ajouter, Retirer

Attributs écrits : File : Ajouter, Retirer
Qcommandée : Retirer
Nbrereçue : Ajouter
Longueurfile : Ajouter, Retirer

Axiomes : C1) **BEG** → filevide(File) = true
C2) **BEG** → Longueurfile = 0
C3) **BEG** → Nbrereçue = 0

C4) Ajouter(x) → **X**File = in(File,x)
C5) Ajouter(x) → **X**Longueurfile = Longueurfile + 1
C6) Ajouter(x) → **X**Nbrereçue = Nbrereçue + 1
C7) Retirer(x) → **X**File = out(File)
C8) Retirer(x) → **X**Qcommandée = x
C9) Retirer(x) → **X**Longueurfile = Longueurfile - 1

C10) Retirer(x) → x = first(File)
C11) Retirer(x) → filevide(File) = false

C12) filevide(File) = true → ∃x **F**ajouter(x)
C13) filevide(File) = false → ∃x **F**Retirer(x)

STOCK

Sorts : NAT, BOOL

Symboles d'opération : +, - : NAT × NAT → NAT
≤ : NAT × NAT → BOOL
0 : → NAT

Symboles d'attribut : Quantité : NAT
Nbresatisfaite : NAT

Symboles d'action : Approvisionner(NAT)
Enlever(NAT)

Attributs lus : Quantité : Approvisionner, Enlever
Nbresatisfaite : Enlever

Attributs écrits : Quantité : Approvisionner, Enlever
Nbresatisfaite : Enlever

Axiomes :

S1) **BEG** → Quantité = 0
S2) **BEG** → Nbresatisfaite = 0

S3) Approvisionner(x) → **X**Quantité = Quantité + x
S4) Enlever(x) → **X**Quantité = Quantité - x
S5) Enlever(x) → **X**Nbresatisfaite = Nbresatisfaite + 1

S6) Enlever(x) → (x ≤ Quantité) = true

S7) (Quantité ≤ 10)=true → ∃x **F**Approvisionner(x)

Par la nouvelle définition de la localité, si l'action "Retirer" de l'objet "COMMANDE" n'est pas exécutée, l'attribut "Qcommandée" ne sera pas modifié. De même, si "Ajouter" ne s'exécute pas, l'attribut 'Nbrereçue" reste inchangé. D'après les axiomes C5 et C9, nous savons qu'il y a une exclusion mutuelle des actions "Retirer" et "Ajouter". Il est donc devenu inutile de préciser que si l'action "Retirer" s'exécute, l'attribut "Qcommandée" reste inchangé et que si l'action "Ajouter" s'exécute, l'attribut "Nbrereçue" reste inchangé.

Pour l'objet "STOCK", la condition de localité nous dit que si "Enlever" n'est pas exécutée, l'attribut "Nbresatisfaite" reste inchangé. D'après les axiomes S3 et S4, il y a une exclusion mutuelle entre les actions "Enlever" et "Approvisionner". Il est donc devenu inutile de préciser que si "Approvisionner" s'exécute, "Nbresatisfaite" reste inchangé.

Les objets "STOCK" et "COMMANDE" sont donc légèrement modifiés. Les axiomes C7, C11 et S6 se révèlent inutiles.

Avec ces nouveaux objets, l'objet "SYSTEME" devient :

SYSTEME

Sorts : NAT, QUEUE, BOOL

Symboles d'opération : filevide : QUEUE → BOOL
in : QUEUE × NAT → QUEUE
out : QUEUE → QUEUE
first : QUEUE → NAT
+, - : NAT × NAT → NAT
≤ : NAT × NAT → BOOL
0 : → NAT

Symboles d'attribut : Réserve : NAT
Cderéalisée : NAT
Liste : QUEUE
Qcdée : NAT
Nbrcde : NAT
Longueurliste : NAT

Symboles d'action : Livrer (NAT)
Enregistrer (NAT)
Stocker (NAT)

Attributs lus : Réserve : Stocker, Livrer
Cderéalisée : Livrer
Liste : Enregistrer, Livrer
Qcdée : ∅
Nbrcde : Enregistrer
Longueurliste : Enregistrer, Livrer

Attributs écrits : Réserve : Stocker, Livrer
Cderéalisée : Livrer
Liste : Enregistrer, Livrer
Qcdée : Livrer
Nbrcde : Enregistrer
Longueurliste : Enregistrer, Livrer

Axiomes :

{Axiomes de "COMMANDE"}

ST1) **BEG** → filevide(Liste) = true
ST2) **BEG** → Longueurliste = 0
ST3) **BEG** → Nbrcde = 0

ST4) Enregistrer(x) → **X**Liste = in(Liste,x)
ST5) Enregistrer(x) → **X**Longueurliste = Longueurliste + 1
ST6) Enregistrer(x) → **X**Nbrcde = Nbrcde + 1
ST7) Livrer(x) → **X**Liste = out(Liste)
ST8) Livrer(x) → **X**Qcdée = x
ST9) Livrer(x) → **X**Longueurliste = Longueurliste - 1

ST10) Livrer(x) → x = first(Liste)
ST11) Livrer(x) → filevide(Liste) = false
ST12) filevide(Liste) = true → ∃ x **F**Enregistrer(x)
ST13) filevide(Liste) = false → ∃ x **F** Livrer(x)

{Axiomes de "STOCK"}

ST14) **BEG** → Réserve = 0
ST15) **BEG** → Cderéalisée = 0

ST16) Stocker(x) → **X**Réserve = Réserve + x
ST17) Livrer(x) → **X**Réserve = Réserve - x
ST18) Livrer(x) → **X**Cderéalisée = Cderéalisée + 1

ST19) Livrer(x) → (x ≤ Réserve) = true

ST20) Réserve = 0 → ∃x **F**Stocker(x)

Avec la nouvelle définition du morphisme, les axiomes qui traduisaient la condition de localité des objets "COMMANDE" et "STOCK" sont devenus inutiles.

Sur ces nouvelles bases, démontrons à nouveau les propriétés de l'exemple 3.1.

Cderéalisée = Nbrcde - Longueurliste

D'après le système de démonstration proposé au chapitre 3 nous devons démontrer :

a) SYSTEME $\Rightarrow$ (**BEG** $\rightarrow$ Cderéalisée = Nbrcde - Longueurliste)
b) SYSTEME $\Rightarrow$ (Cderéalisée = Nbrcde - Longueurliste $\rightarrow$
X (Cderéalisée = Nbrcde - Longueurliste))

Le point a) se démontre de façon triviale par les axiomes ST2, ST3 et ST15.
Pour démontrer b) il faut isoler tous les cas, ce qui nous donne :

b.1) Livrer(x) $\rightarrow$ (Cderéalisée = Nbrcde - Longueurliste $\rightarrow$ **X** (Cderéalisée = Nbrcde - Longueurliste))

b.2) Enregistrer(x) $\rightarrow$ (Cderéalisée = Nbrcde - Longueurliste $\rightarrow$
X (Cderéalisée = Nbrcde - Longueurliste))

b.3) Stocker(x) $\rightarrow$ (Cderéalisée = Nbrcde - Longueurliste $\rightarrow$ **X** (Cderéalisée = Nbrcde - Longueurliste))

b.4) $\neg\exists$x (Livrer(x) $\vee$ Enregistrer(x) $\vee$ Stocker(x)) $\rightarrow$ (Cderéalisée = Nbrcde - Longueurliste $\rightarrow$
X (Cderéalisée = Nbrcde - Longueurliste))

La démonstration de b.1) est :

1. Livrer(x) $\rightarrow$ **X**Longueurliste = Longueurliste - 1	ST9
2. Enregistrer(y) $\rightarrow$ **X**Longueurliste = Longueurliste + 1	ST5
3. Livrer(x) $\rightarrow \neg$Enregistrer(y)	1. et 2.
4. $\neg$Enregistrer(y) $\rightarrow$ **X**Nbrcde = Nbrcde	condition de localité
5. Livrer(x) $\rightarrow$ **X**Nbrcde = Nbrcde	3. et 4.
6. Livrer(x) $\rightarrow$ **X**Cderéalisée = Cderéalisée + 1	ST18
7. Livrer(x) $\rightarrow$ (Cderéalisée = Nbrcde - Longueurliste $\rightarrow$ X (Cderéalisée = Nbrcde - Longueurliste))	**1., 5. et 6.**

La démonstration de b.2) se fait de la manière suivante : prenons les différents cas possibles et observons ce qui se passe.
Quatre cas peuvent se présenter :
1°) les actions "Enregistrer", "Livrer" et "Stocker" se déroulent en même temps;
2°) les actions "Enregistrer" et "Livrer" se déroulent en même temps;
3°) les actions "Enregistrer" et "Stocker" se déroulent en même temps;
4°) seule l'action "Enregistrer" se déroule.

Nous avons déjà démontré que si l'action "Livrer" se déroulait, l'invariant restait inchangé. Voyons ce qui se passe lorsqu'elle ne s'exécute pas.

1. Enregistrer(x) → **X**Longueurliste = Longueurliste + 1	ST5
2. ¬Livrer(y) → **X**Cderéalisée = Cderéalisée	condition de localité
3. Enregistrer(x) → **X**Nbrcde = Nbrcde + 1	ST6
4. Enregistrer(x) ∧ ¬∃z Livrer(y) → (Cderéalisée = Nbrcde - Longueurliste → **X**(Cderéalisée = Nbrcde - Longueurliste))	1., 2. et 3.

De ces différents cas, nous pouvons déduire :

5. Enregistrer(x) → (Cderéalisée = Nbrcde - Longueurliste → X(Cderéalisée = Nbrcde - Longueurliste))

Nous avons démontré dans b.1) et b.2) que si nous avons l'invariant à un moment donné et si "Livrer" ou "Enregistrer" s'exécute, l'invariant sera toujours respecté à l'instant suivant. Pour démontrer b.3), il nous reste à voir ce qui se passe lorsque seule l'action "Stocker" s'exécute.

1. ¬Livrer(x) → **X**Cderéalisée = Cderéalisée	condition de localité
2. ¬∃x (Enregistrer(x) ∨ Livrer(x)) → **X**Nbrcde = Nbrcde ∧ **X**Longueurliste = Longueurliste	condition de localité

3. Stocker(y) ∧ ¬∃x Enregistrer(x) ∧ ¬∃z Livrer(z) → (Cderéalisée = Nbrcde - Longueurliste → X(Cderéalisée = Nbrcde - Longueurliste)) **1. et 2.**

La démonstration de b.4) est triviale car par la condition de localité, nous savons que si les actions "Livrer", "Stocker" et "Enregistrer" ne s'exécutent pas, les attributs "Cderéalisée", "Nbrcde" et "Longueurliste" restent inchangés.

5.3.3. Conditions sur l'abstraction

Grâce au concept des attributs lus et modifiés, nous pourrons assurer une meilleure gestion de la concurrence. Dans la signature, nous avons défini les ensembles inverses. Pour pouvoir travailler sur les ensembles des attributs que peut lire ou écrire une action, nous devons les définir.

Pour une signature d'objet $\theta = (\Sigma, A, \Gamma)$ où :

⇨ $A^{\bullet} = \{att_1,..., att_n\}$

⇨ $\Gamma^{\bullet} = \{act_1,..., act_m\}$

on a:

$\forall\ act_i \in \Gamma^{\bullet} : \Lambda(act_i) = \{att_{i_1}, ..., att_{i_p}\}$ tel que $\forall\ 1 \leq j \leq p\ act_i \in \Theta\ (att_{i_1})$

att_{i_j} sera le j^{ème} attribut pouvant être lu par act_i ,
$\forall\ act_i \in \Gamma^{\bullet} : M(act_i) = \{att_{i_1}, ..., att_{i_p}\}$ tel que $\forall\ 1 \leq j \leq p\ act_i \in \Xi\ (att_{i_1})$
att_{i_j} sera le j^{ème} attribut pouvant être modifié par act_i ,

avec $1 \leq i \leq m$, $1 \leq i_1 < ... < i_p \leq n$ et $1 \leq p \leq n$.

Formalisons cette nouvelle approche en respectant la même structure que dans le chapitre 3. Les ensembles définis nous seront utiles mais il nous faut introduire de nouveaux éléments :

1) une fonction $\beta : \Gamma_A^{\bullet} \rightarrow P\ (\Gamma_C^{\bullet})$ qui associe à chaque action de "A" l'ensemble des actions concrètes qui l'implémentent;
2) l'ensemble $\Lambda_{\beta(a)}$ qui est l'ensemble des attributs lus par les actions implémentant l'action "a";
3) l'ensemble $M_{\beta(a)}$ qui est l'ensemble des attributs modifiés par les actions implémentant l'action "a";

Reprenons les structures d'interprétations $I_A=(U_A,A_A,G_A)$ pour l'objet abstrait "A" et $I_C=(U_C,A_C,G_C)$ pour le corps "C" et définissons à nouveau l'abstraction. Les modifications que nous allons apportées à la définition donnée au chapitre 3 portent sur les points 1)c) , 2)c) et 2)d).

1) Une fonction $\rho : w \rightarrow 2^w$ (du temps abstrait vers le temps concret)
telle que pour chaque $i \in w$,
a) $\rho(i)$ est un intervalle $[beg(i),end(i)]$;
b) $beg(i) < beg(i+1)$ et $end(i) < end(i+1)$;
c) pour tout $a,b \in \Gamma_A^{\bullet}$ et $i \in G_A(a)$ et $i+1 \in G_A(b)$: on a que
pour tout m : $beg(i+1) \leq m < end(i)$ et
tout attribut $f \in (A_N^{\bullet} \cap ((\Lambda_{\beta(a)} \cap M_{\beta(b)}) \cup (M_{\beta(a)} \cap \Lambda_{\beta(b)}) \cup (M_{\beta(a)} \cap M_{\beta(b)})))$,
$A_C(f)(m)=A_C(f)(m+1)$;
(lors de la superposition des intervalles, les attributs qui sont partagés, lus ou écrits, par les actions abstraites de chacun des intervalles ne peuvent être modifiés).

2) Une fonction $G_A : \Gamma_A^{\bullet} \rightarrow P_A(N_0)$ (fonction allant des actions abstraites vers le temps abstrait, qui associe à toutes les actions abstraites les moments où l'action se produit) telle que,
pour tout $a \in \Gamma_A^{\bullet}$ et $i \in G_A(a)$, il existe exactement un $j \in \rho(i)$ et un $k \in \rho(i)$, $j \leq k$ tels que
a) $j \in G_C(beg_a)$;
b) $k \in G_C(end_a)$;

c) $A_C(f)(m) = A_C(f)(m+1)$ pour tout attribut $f \in (A_N^{\bullet} \cap \Lambda_{\beta(a)})$ et $beg(i) \leq m < j$
(aucun attribut lu par l'action ne peut être modifié entre le début de l'instant et le début de l'implémentation de l'action);
d) $A_C(f)(m) = A_C(f)(m+1)$ pour tout attribut $f \in (A_N^{\bullet} \cap M_{\beta(a)})$ et $k < m \leq end(i)$
(Après la fin de l'action, aucun attribut modifié par l'implémentation de l'action ne peut changer jusqu'à la fin de l'intervalle).

5.4. Développement d'un exemple

5.4.1. Description de l'objet abstrait.

Exemple 5.2.

L'exemple choisi est celui d'un objet dont le but est de permuter deux variables de types quelconques.
Sa description est la suivante :

A

Sorts : T

Symboles d'attribut : A, B : T

Symboles d'action : Permuter

Attributs lus : A : Permuter
B : Permuter

Attributs écrits : A : Permuter
B : Permuter

Axiomes : A1) Permuter $\rightarrow$ (**X**A = B) $\wedge$ (**X**B = A)

5.4.2. Description de l'objet concret

Dans l'objet abstrait, l'action "Permuter" est considérée comme une opération atomique. Dans un niveau plus concret, il faudra copier la valeur d'une des variables dans un buffer afin que celle-ci ne soit pas écrasée. Il apparait que, contrairement à l'exemple du chapitre 4, l'action abstraite n'est pas implémentée grâce à une boucle mais en réalisant trois actions dans un ordre

déterminé. Pour réaliser cela, on aura dans notre objet concret un nouvel attribut appelé "Buffer" dont le type sera le même que celui des attributs "A" et "B".

N

Sorts : T

Symboles d'attribut :A, B, Buffer : T

Symboles d'action : DonnervaleurA
DonnervaleurB
DonnervaleurBuffer

Attributs lus : A : DonnervaleurBuffer
B : Donner valeurA
Buffer : DonnervaleurB

Attributs écrits : A : DonnervaleurA
B : DonnervaleurB
Buffer : DonnervaleurBuffer

Axiomes : N1) DonnervaleurA → **X**A = B
N2) DonnervaleurB → **X**B = Buffer
N3) DonnervaleurBuffer → **X**Buffer = A

5.4.3. Construction du corps

Afin de réaliser les trois actions de l'objet concret dans l'ordre voulu, introduisons un objet "ETAT" dont le but sera d'indiquer l'état d'avancement de l'action "Permuter".

ETAT

Sorts : NAT

Symboles d'opération : 0, 1, 2, 3 : → NAT

Symboles d'attribut : Etat : NAT

Symboles d'action : Initialiser
Changement1
Changement2
Changement3

Attributs lus :	Etat : Changement1, Changement2, Changement3
Attributs écrits :	Etat : Initialiser, Changement1, Changement2, Changement3
Axiomes :	E1) Initialiser → **X**Etat = 0
	E2) Changement1 → **X**Etat = 1
	E3) Changement2 → **X**Etat = 2
	E4) Changement3 → **X**Etat = 3
	E5) Changement1 → Etat = 0
	E6) Changement2 → Etat = 1
	E7) Changement3 → Etat = 2
	E8) Etat=0 → **F**Changement1
	E9) Etat=1 → **F**Changement2
	E10) Etat=2 → **F**Changement3

Nous avons un attribut "Etat" qui peut prendre les valeurs 0, 1, 2 ou 3. Il peut être modifié par quatre actions synchronisées avec le début de l'action "Permuter" et les trois actions de l'objet concret.

Plus généralement, lorsque une action abstraite est implémentée par n actions concrètes act_1, ..., act_n qui doivent être exécutées dans cet ordre, nous construisons un objet. qui possèdera un attribut "Etat" pouvant prendre les valeurs 1 à n.
Cet objet a une action "Initialiser" et n actions "Changement1", ..., "Changementn". Elles seront synchronisées avec le début de l'action abstraite et les n actions concrètes qui implémentent cette action.

Au niveau des axiomes, pour i allant de 1 à n, nous aurons :
Initialiser → **X**Etat = 0
Changementi → **X**Etat = i
Changementi → Etat = i - 1
Etat = i - 1 → **F**Changement

Ensuite, nous devons construire la structure de conception pour N. Elle est décrite à la figure 5.1.

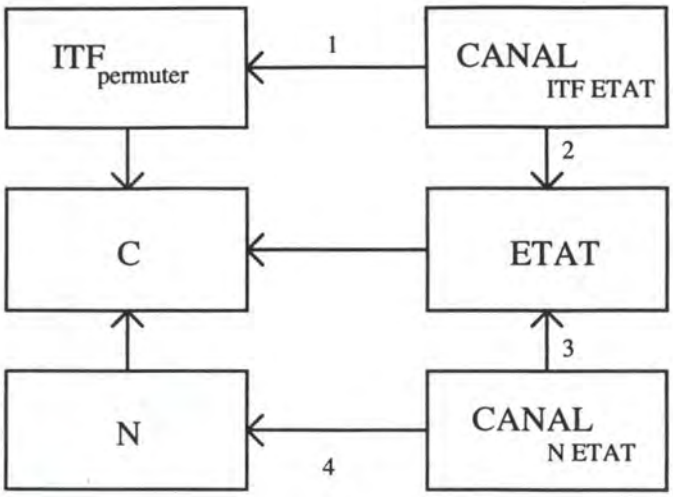

Fig. 5.1. *: Structure de conception pour N.*

L'objet $ITF_{Permuter}$ est le suivant :

ITFPermuter

Sorts : BOOL

Symboles d'opération : /

Symboles d'attribut : $In_{Permuter}$: BOOL

Symboles d'action : $Beg_{Permuter}$
$End_{Permuter}$

Attributs lus : $In_{Permuter}$: $Beg_{Permuter}$, $End_{Permuter}$

Attributs écrits : $In_{Permuter}$: $Beg_{Permuter}$, $End_{Permuter}$

Axiomes : I1) $\mathbf{BEG} \rightarrow \neg\, In_{Permuter}$
I2) $Beg_{Permuter} \rightarrow \neg\, In_{Permuter} \wedge (\mathbf{X} In_{Permuter} \vee End_{Permuter})$
I3) $End_{Permuter} \rightarrow (In_{Permuter} \vee Beg_{Permuter}) \wedge \mathbf{X} \neg\, In_{Permuter}$

Nous aurons les canaux :

CANAL ITF ETAT

Sorts : /

Symboles d'opération : /

Symboles d'attribut : /

Symboles d'action : Action0

CANAL N ETAT

Sorts : /

Symboles d'opération : /

Symboles d'attribut : /

Symboles d'action : Action1
Action2
Action3

Les différents composants de cette structure vont être reliés par les morphismes 1, 2, 3 et 4. Leur descritption est la suivante :

1

Relation : CANAL $_{\text{ITF ETAT}} \rightarrow \text{ITF}_{\text{permuter}}$
sorts : /
Symboles d'action : Action0 $\rightarrow \text{Beg}_{\text{permuter}}$

2

Relation : CANAL $_{ITF\ ETAT}$ → ETAT
sorts : /
Symboles d'action : Action0 → Initialiser

3

Relation : CANAL $_{N\ ETAT}$ → ETAT
sorts : /
Symboles d'action : Action1 → Changement1
Action2 → Changement2
Action3 → Changement3

4

Relation : CANAL $_{N\ ETAT}$ → N
sorts : /
Symboles d'action : Action1 → DonnervaleurBuffer
Action2 → DonnervaleurB
Action3 → DonnervaleurA

Pour construire le corps, il nous reste à faire la somme des composants de la structure de conception et d'ajouter des axiomes qui implémentent la localité de l'objet abstrait et les axiomes de fin.

Ces derniers ont la forme suivante :

$Etat = n \wedge In_{action} \rightarrow \mathbf{F}End_{action}$

$End_{action} \rightarrow Etat = n \wedge In_{action}$

C

Sorts : NAT, BOOL

Symboles d'opération : 0, 1, 2, 3 : → NAT

Symboles d'attribut : A, B, Buffer : NAT
Etat : NAT
$In_{Permuter}$: BOOL

Symboles d'action : $Beg_{Permuter}$
$End_{Permuter}$
DonnervaleurA
DonnervaleurB
DonnervaleurBuffer

Attributs lus : A : DonnervaleurBuffer
B : Donner valeurA
Buffer : DonnervaleurB
Etat : DonnervaleurA, DonnervaleurB, DonnervaleurBuffer
$In_{Permuter}$: $Beg_{Permuter}$, $End_{Permuter}$

Attributs écrits : A : DonnervaleurA
B : DonnervaleurB
Buffer : DonnervaleurBuffer
Etat : $Beg_{Permuter}$, DonnervaleurA, DonnervaleurB, DonnervaleurBuffer
$In_{Permuter}$: $Beg_{Permuter}$, $End_{Permuter}$

Axiomes :

{ Axiomes de "N". }

C1) DonnervaleurA → **X**A = B
C2) DonnervaleurB → **X**B = Buffer
C3) DonnervaleurBuffer → **X**Buffer = A

{ Axiomes de "ETAT". }

C4) $Beg_{Permuter}$ → **X**Etat = 0
C5) DonnervaleurBuffer → **X**Etat = 1
C6) DonnervaleurA → **X**Etat = 2
C7) DonnervaleurB → **X**Etat = 3
C8) Etat = 0 → **F**DonnervaleurBuffer
C9) Etat = 1 → **F**DonnervaleurA
C10) Etat = 2 → **F**DonnervaleurB
C11) DonnervaleurBuffer → Etat = 0

C12) DonnervaleurA $\rightarrow$ Etat = 1
C13) DonnervaleurB $\rightarrow$ Etat = 2

{ Axiomes de l'interface "ITF$_{Permuter}$". }

C14) **BEG** $\rightarrow \neg In_{Permuter}$
C15) $Beg_{Permuter} \rightarrow \neg In_{Permuter} \wedge (\mathbf{X}In_{Permuter} \vee End_{Permuter})$
C16) $End_{Permuter} \rightarrow (In_{Permuter} \vee Beg_{Permuter}) \wedge \mathbf{X}\neg In_{Permuter}$

{ Axiomes qui implémentent la localité de "A". Ils empêchent les actions "DonnervaleurA" et "DonnervaleurB" de s'exécuter si on n'est pas à l'intérieur de l'action permuter. Les attributs "A" et "B" ne seront donc pas modifiés si on ne se trouve pas à l'intérieur de cette action, ce qui est bien la condition de localité de l'objet abstrait "A" }

C17) $DonnervaleurBuffer \rightarrow In_{Permuter}$
C18) $DonnervaleurA \rightarrow In_{Permuter}$
C19) $DonnervaleurB \rightarrow In_{Permuter}$

{ Axiomes de fin. }

C20) Etat = 3 $\wedge$ $In_{Permuter} \rightarrow \mathbf{F}End_{Permuter}$
C21) $End_{Permuter} \rightarrow In_{Permuter} \wedge$ Etat = 3

5.4.4. Preuve des propriétés de "A"

Règles de transition

Nous avons vu dans le chapitre 4 que les conditions sur l'abstraction et la définition de l'A-interprétation étaient importantes pour démontrer la correction des règles de transition. Or, les conditions 2c) et 2d) ont été modifiées par l'apparition des ensembles Θ et Ξ. Pour que les règles restent cohérentes, nous imposons certaines contraintes sur les conditions ψ et φ.

Dans la règle de changement, la condition φ est une formule du premier ordre sur les attributs de l'objet "A" que l'action "a" peut lire. La condition ψ est une formule du premier ordre sur les attributs de "A" que l'action "a" peut modifier.

Dans les règles de Sécurité et de Vivacité, la condition φ est une formule du premier ordre sur les attributs de l'objet "A" et que l'action "a" peut lire.

Dans la règle de Début, la condition φ est une formule du premier ordre sur les attributs de l'objet "A".

La démonstration de la cohérence de ces règles se fera de la même manière qu'au chapitre 4. Les modifications des conditions 2c) et 2) sur l'abstraction nous obligent cependant à changer certaines parties de la preuve. Nous ne reprenons ici que les parties modifiées.

1• Changement.

2. La condition φ est une formule du premier ordre sur les attributs de l'objet "A" que l'action "a" peut lire. D'après la condition 2c), nous savons que les attributs lus par l'action "a" ont les mêmes valeurs au début de l'action qu'au début de l'intervalle, c'est à dire à l'instant $beg(\rho(i))$. Nous avons donc :
si $j \in \rho(i)$ et $j \in G(beg_a)$
$A_C(\iota(\varphi))(j) = A_C(\iota(\varphi))(beg(\rho(i)))$.
De la définition de l'A-interprétation, nous savons que
$A_{(\rho, G_A)}(\varphi)(i) = A_C(\iota(\varphi))(beg(\rho(i)))$.
En effet, φ est une formule du premier ordre sur les attributs de l'objet abstrait.
Par transition, nous obtenons :
$A_C(\iota(\varphi))(j) = A_{(\rho, G_A)}(\varphi)(i)$.
Si nous avons la formule φ à l'instant i où "a" s'exécute, nous aurons $\iota(\varphi)$ lorsque "beg_a" s'exécutera durant l'intervalle $\rho(i)$.

3. Pour démontrer que $A_C(\iota(\varphi))(beg(\rho(i + 1))) = A_C(\iota(\varphi))(j + 1)$ avec $j \in \rho(i)$ et $j \in G(end_a)$ nous allons procéder de la même manière qu'au chapitre 4.

A. Intervalles disjoints.

La démonstration du lemme ne change pas, nous aurons donc que les attributs du noyau ne changent pas dans l'intervalle $]end(i),beg(i+1)[$.

La condition ψ est une formule du premier ordre sur les attributs de "A" que l'action "a" peut modifier. 2d) nous assure qu'après l'exécution de end_a, les valeurs des attributs que l'action "a" peut modifier restent inchangées jusqu'à la fin de l'intervalle. Nous pouvons dire que :
$A_C(\iota(\varphi))(j + 1) = A_C(\iota(\varphi))(end(\rho(i))$ avec $j \in \rho(i)$ et $j \in G(end_a)$.
D'après le lemme, nous avons $A_C(\iota(\varphi))(beg(\rho(i + 1)) = A_C(\iota(\varphi))(end(\rho(i))$
Par transition :
$A_C(\iota(\varphi))(beg(\rho(i + 1)) = A_C(\iota(\varphi))(j + 1)$

B. Intervalles superposés.

D'après la condition 1c), aucun attribut du noyau ne peut être modifié dans l'intervalle.
$A_C(\iota(\varphi))(m) = A_C(\iota(\varphi))(m + 1)$ pour tout $beg(\rho(i + 1)) \leq m < end(\rho(i))$.
Nous aurons donc $A_C(\iota(\varphi))(beg(\rho(i + 1)) = A_C(\iota(\varphi))(end(\rho(i)))$.
La condition 2d) nous dit que :
$A_C(\iota(\varphi))(j + 1) = A_C(\iota(\varphi))(end(\rho(i))$ avec $j \in \rho(i)$ et $j \in G(end_a)$.
Par transition, nous pouvons déduire que :
$A_C(\iota(\varphi))(beg(\rho(i + 1)) = A_C(\iota(\varphi))(j + 1)$.

2• Sécurité, 3• Vivacité et 4• Début

Ces démonstrations sont identiques à celles faites au chapitre 4.

Démonstration de la propriété

On doit démontrer que notre objet C a le même comportement que l'objet abstrait A. Pour cela, démontrons que l'objet C a bien la propriété suivante :

1. Permuter $\rightarrow$ $(XA = B) \wedge (XB = A)$

Utilisons la règle de changement vue au chapitre 4.

$$C \Rightarrow (Beg_a \wedge \iota(\varphi) \rightarrow \mathbf{X}In_a \, \mathcal{u} \, (End_a \wedge \mathbf{X}\iota(\psi)))$$

1• changement : $\dfrac{}{A \Rightarrow (a \wedge \varphi \rightarrow \mathbf{X}\psi)}$

D'après cette règle, pour démontrer la propriété 1., il suffit de démontrer que

$C \Rightarrow (Beg_{Permuter} \wedge (A = A_0 \wedge B = B_0) \rightarrow XIn_{Permuter} \, \mathcal{u} \, (End_{Permuter} \wedge XB = A_0 \wedge XA = B_0))$

Comment prouver cela ?

D'après la définition de l'opérateur temporel $\mathcal{u}$, il nous faut démontrer deux choses :

1. $Beg_{Permuter} \wedge (A = A_0 \wedge B = B_0) \rightarrow \mathbf{F}(End_{Permuter} \wedge \mathbf{X}B = A_0 \wedge \mathbf{X}A = B_0)$
2. $Beg_{Permuter} \wedge (A = A_0 \wedge B = B_0) \rightarrow \mathbf{X}In_{Permuter} \, \mathcal{w} \, (End_{Permuter} \wedge \mathbf{X}B = A_0 \wedge \mathbf{X}A = B_0)$.

1.A. Correction partielle

$$Beg_{Permuter} \wedge (A = A_0 \wedge B = B_0) \rightarrow (End_{Permuter} \rightarrow XB = A_0 \wedge XA = B_0)$$

Lorsque nous avons des actions qui s'exécutent de manière séquentielle, pour démontrer qu'une propriété est respectée, il est plus aisé de voir ce qui se passe à chaque étape.

Si nous projetons l'exécution de l'action "Permuter" sur la ligne de temps concret, nous aurons :

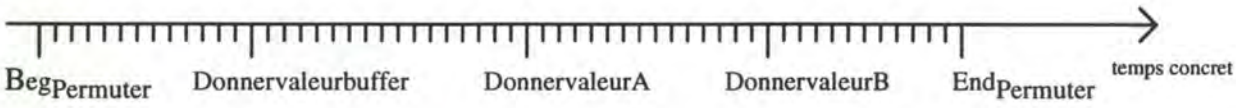

Fig. 5.2. *: Projection de l'action "Permuter" sur la ligne de temps concret.*

Nous devons observer ce qui se passe dans l'intervalle [$Beg_{Permuter}$ .. $End_{Permuter}$] et montrer que nous avons bien la propriété. Nous devons donc voir ce qui se passe à chaque instant concret qui est compris dans cet intervalle. D'après la preuve réalisée au point 5.2., on sait que "$In_{Permuter}$" est vrai dans l'intervalle]$Beg_{Permuter}$ .. $End_{Permuter}$]. Il nous faut donc démontrer les neuf points suivants :

A1) $Beg_{Permuter} \wedge A = A_0 \wedge B = B_0 \rightarrow \mathbf{X}Etat = 0 \wedge \mathbf{X}A = A_0 \wedge \mathbf{X}B = B_0$

A2) $In_{Permuter} \wedge Etat = 0 \wedge \neg DonnervaleurBuffer \wedge A = A_0 \wedge B = B_0 \rightarrow \mathbf{X}Etat = 0 \wedge \mathbf{X}A = A_0 \wedge \mathbf{X}B = B_0$

A3) $In_{Permuter} \wedge Etat = 0 \wedge DonnervaleurBuffer \wedge A = A_0 \wedge B = B_0 \rightarrow \mathbf{X}Etat = 1 \wedge \mathbf{X}A = A_0 \wedge \mathbf{X}B = B_0 \wedge \mathbf{X}Buffer = A_0$

A4) $In_{Permuter} \wedge Etat = 1 \wedge \neg DonnervaleurA \wedge A = A_0 \wedge B = B_0 \wedge Buffer = A_0 \rightarrow \mathbf{X}Etat = 1 \wedge \mathbf{X}A = A_0 \wedge \mathbf{X}B = B_0 \wedge \mathbf{X}Buffer = A_0$

A5) $In_{Permuter} \wedge Etat = 1 \wedge DonnervaleurA \wedge A = A_0 \wedge B = B_0 \wedge Buffer = A_0 \rightarrow \mathbf{X}Etat = 2 \wedge \mathbf{X}A = B_0 \wedge \mathbf{X}B = B_0 \wedge \mathbf{X}Buffer = A_0$

A6) $In_{Permuter} \wedge Etat = 2 \wedge \neg DonnervaleurB \wedge A = B_0 \wedge B = B_0 \wedge Buffer = A_0 \rightarrow \mathbf{X}Etat = 2 \wedge \mathbf{X}A = B_0 \wedge \mathbf{X}B = B_0 \wedge \mathbf{X}Buffer = A_0$

A7) $In_{Permuter} \wedge Etat = 2 \wedge DonnervaleurB \wedge A = B_0 \wedge B = B_0 \wedge Buffer = A_0 \rightarrow \mathbf{X}Etat = 3 \wedge \mathbf{X}A = B_0 \wedge \mathbf{X}B = A_0 \wedge \mathbf{X}Buffer = A_0$

A8) $In_{Permuter} \wedge Etat = 3 \wedge \neg End_{Permuter} \wedge A = B_0 \wedge B = A_0 \wedge Buffer = A_0 \rightarrow \mathbf{X}Etat = 3 \wedge \mathbf{X}A = B_0 \wedge \mathbf{X}B = A_0 \wedge \mathbf{X}Buffer = A_0$

A9) $In_{Permuter} \wedge Etat = 3 \wedge End_{Permuter} \wedge A = B_0 \wedge B = A_0 \wedge Buffer = A_0 \rightarrow \mathbf{X}A = B_0 \wedge \mathbf{X}B = A_0$

1.B. Correction totale

$$Beg_{Permuter} \rightarrow \mathbf{F}End_{Permuter}$$

Détails de la preuve

A1.	1. $Beg_{Permuter} \rightarrow \mathbf{X}Etat=0$	C4
	2. $Beg_{Permuter} \rightarrow \neg In_{Permuter} \wedge (\mathbf{X}In_{Permuter} \vee End_{Permuter})$	C15
	3. DonnervaleurA $\rightarrow In_{Permuter}$	C18
	4. $\neg In_{Permuter} \rightarrow \neg$DonnervaleurA	3. et P.C.
	5. $Beg_{Permuter} \rightarrow \neg$DonnervaleurA	2., 4. et P.C.
	6. DonnervaleurB $\rightarrow In_{Permuter}$	C19
	7. $\neg In_{Permuter} \rightarrow \neg$DonnervaleurB	6. et P.C.
	8. $Beg_{Permuter} \rightarrow \neg$DonnervaleurB	2., 7. et P.C.
	9. $\neg$DonnervaleurA $\rightarrow \mathbf{X}A = A$	condition de localité
	10. $\neg$DonnervaleurB $\rightarrow \mathbf{X}B = B$	condition de localité
	11. $Beg_{Permuter} \wedge A = A_0 \wedge B = B_0 \rightarrow \mathbf{X}Etat = 0 \wedge \mathbf{X}A = A_0 \wedge \mathbf{X}B = B_0$	**1., 5., 8., 9. et 10.**
A2.	12. DonnervaleurA $\rightarrow$ Etat = 1	C12
	13. $\neg$(Etat = 1) $\rightarrow \neg$DonnervaleurA	12. et P.C.
	14. DonnervaleurB $\rightarrow$ Etat = 2	C13
	15. $\neg$(Etat = 2) $\rightarrow \neg$DonnervaleurB	14. et P.C.
	16. Etat = 0 $\rightarrow \neg$(DonnervaleurA $\vee$ DonnervaleurB)	13., 15. et P.C.
	17. Etat=0 $\rightarrow \mathbf{X}A = A \wedge \mathbf{X}B = B$	9., 10. et 16.
	18. $Beg_{Permuter} \rightarrow \neg In_{Permuter} \wedge (\mathbf{X}In_{Permuter} \vee End_{Permuter})$	C16
	19. $In_{Permuter} \rightarrow \neg Beg_{Permuter}$	18. et P.C.
	20. $\neg$ ($Beg_{Permuter} \vee$ DonnervaleurBuffer $\vee$ DonnervaleurA $\vee$ DonnervaleurB) $\rightarrow \mathbf{X}Etat = Etat$	condition de localité
	21. $In_{Permuter} \wedge Etat = 0 \wedge \neg$DonnervaleurBuffer $\wedge A = A_0 \wedge B = B_0 \rightarrow \mathbf{X}Etat = 0 \wedge \mathbf{X}A = A_0 \wedge \mathbf{X}B = B_0$	**17. et 20.**
A3.	22. DonnervaleurBuffer $\rightarrow \mathbf{X}Etat = 1$	C5
	23. DonnervaleurBuffer $\rightarrow \mathbf{X}$Buffer = A	C3
	24. $In_{Permuter} \wedge Etat = 0 \wedge$ DonnervaleurBuffer $\wedge A = A_0 \wedge B = B_0 \rightarrow \mathbf{X}Etat = 1 \wedge \mathbf{X}A = A_0 \wedge \mathbf{X}B = B_0 \wedge \mathbf{X}$Buffer $= A_0$	**17.,22. et 23.**

Les autres étapes de la démonstration se prouvent de manière similaire.

Prouvons maintenant la correction totale.

Pour démontrer que l'action abstraite se termine, il faut prouver que tôt ou tard l'attribut "Etat" prendra la valeur 3, ce qui implique que l'action est réalisée.

1. $Beg_{Permuter} \rightarrow \mathbf{X}Etat = 0$	C4
2. Etat = 0 $\rightarrow \mathbf{F}$DonnervaleurBuffer	C8
3. DonnervaleurBuffer $\rightarrow \mathbf{X}Etat = 1$	C5
4. Etat = 1 $\rightarrow \mathbf{F}$DonnervaleurA	C9
5. DonnervaleurA $\rightarrow \mathbf{X}Etat = 2$	C6
6. Etat = 2 $\rightarrow \mathbf{F}$DonnervaleurB	C10
7. DonnervaleurB $\rightarrow \mathbf{X}Etat = 3$	C7
8. Etat = 3 $\wedge \neg Beg_{Permuter} \rightarrow \mathbf{F}End_{Permuter}$	C20

Nous pouvons démontrer la correction totale en utilisant ces huit points et les règles suivantes :

a. $\dfrac{\varphi \rightarrow X\psi}{\varphi \rightarrow F\psi}$

b. $\dfrac{A \rightarrow FB,\ B \rightarrow FC}{A \rightarrow FC}$

2. D'après les preuves faites au point 5.2., nous savons que nous pouvons déduire "$In_{Permuter} \rightarrow \mathbf{X}In_{Permuter}\ \mathcal{W}\ End_{Permuter}$" de l'interface de "Permuter". Ceux-ci disent aussi que "$Beg_{Permuter} \rightarrow \neg\ In_{Permuter} \wedge (\mathbf{X}In_{Permuter} \vee End_{Permuter})$". De cela, nous déduisons que "$Beg_{Permuter} \rightarrow End_{Permuter} \vee \mathbf{X}In_{Permuter}\ \mathcal{W}\ End_{Permuter}$". Nous avons donc "$Beg_{Permuter} \rightarrow \mathbf{X}In_{Permuter}\ \mathcal{W}\ End_{Permuter}$".
A fortiori, nous avons "$Beg_{Permuter} \wedge (A = A_0 \wedge B = B_0) \rightarrow \mathbf{X}In_{Permuter}\ \mathcal{W}\ End_{Permuter}$"
Nous avons démontré au point A. que si on commençait l'action "Permuter" avec $A = A_0 \wedge B = B_0$, tôt ou tard, celle-ci se terminera et nous aurons $\mathbf{X}B = A_0 \wedge \mathbf{X}A = B_0$.
Nous avons donc démontré que "$Beg_{Permuter} \wedge (A = A_0 \wedge B = B_0) \rightarrow \mathbf{X}In_{Permuter}\ \mathcal{W}$ $(End_{Permuter} \wedge \mathbf{X}B = A_0 \wedge \mathbf{X}A = B_0)$".

Chapitre 6

Méthode générale d'implémentation d'un objet abstrait en termes d'un objet concret

6.1. Introduction

Dans les chapitres précédents, nous avons développé des exemples qui illustraient la théorie de J.L. Fiadeiro et de T. Maibaum. Nous avons vu comment implémenter des objets abstraits en terme d'objets concrets grâce à une boucle ou à l'exécution séquentielle de plusieurs actions. De manière intuitive, nous avons construit des objets, synchronisé leurs actions et ajouté des axiomes de manière à réaliser ces types de structures pour que l'objet concret ait le même comportement que l'objet abstrait.

En s'inspirant de ces exemples, une méthode est développée dans ce chapitre dont le but est de construire de manière automatique tout programme. Cette construction est basée sur la méthode des raffinements successifs. Partant d'un objet de départ dont le comportement est décrit par des actions abstraites, des modifications successives lui seront apportées et cela, jusqu'à l'obtention d'un objet qui puisse être implémenté dans l'environnement désiré. A chaque étape, nous implémenterons les actions de l'objet abstrait en termes d'un objet de granularité inférieure.

L'implémentation d'une action abstraite peut être réalisée par différentes structures de programme : la séquence, la boucle et le "if ... then ... else ...". Dans un premier temps, nous

abordons la construction du corps à partir de l'objet concret pour que celui-ci implémente une action abstraite. Cette construction est analysée indépendamment des autres actions. Dans un deuxième temps, nous aborderons la gestion des interactions entre ces différentes actions. A partir de là, nous dégagerons une méthode générale d'implémentation d'un objet abstrait en termes d'un objet concret.

6.2. Implémentation d'une action en termes d'un objet concret

Comment implémenter une action nécessitant la réalisation d'une séquence, d'une boucle ou d'un "if ... then ... else ..." ?

Notation : Dans cette partie, l'action que nous implémentons est nommée "A".

6.2.1. Réalisation d'une séquence

Pour représenter la séquence, et afin que celle-ci se déroule dans le bon ordre, nous devons introduire quelque chose qui nous renseigne sur son état d'avancement. Nous pourrions introduire un attribut dans le noyau mais, nous trouvant dans une optique orientée objet, il est préférable de créer un objet : objet que nous appellons "ETAT". Il comporte un attribut et plusieurs actions que nous synchronisons avec les actions de l'interface et les actions à exécuter se trouvant dans le noyau. Celui-ci peut être constitué d'une collection d'objets. Dans ce cas, nous supposerons qu'il existe une étape préliminaire au cours de laquelle nous construirons le noyau comme la colimite de la structure de conception réalisée sur l'ensemble de ces objets. L'hypothèse posée tout au long de ce chapitre est que le noyau est un objet unique à l'intérieur duquel toutes les actions qui devaient être synchronisées le sont déjà. Cela nous permet de laisser tomber certains problèmes posés par la synchronisation de "ETAT" avec une collection d'objets.

6.2.1.1. Construction de l'objet "ETAT"

De manière générale, une séquence peut comprendre m actions différentes "$Action_1$", ..., "$Action_m$" qui devront s'exécuter dans un certain ordre. Une même action peut se répéter à plusieures reprises. Dans ce cas, nous construisons l'objet "ETAT" de la manière suivante : il comporte un attribut "Etat", qui pourra prendre les valeurs 1 à n, m actions nommées "$Changement_1$", ..., "$Changement_m$" et l'action "Initialiser". L'attribut "Etat" pourra être lu et modifié par les actions "Initialiser", "$Changement_1$", ... et "$Changement_m$". La séquence sera dès lors constituée de m actions différentes et n étapes.

Quant aux axiomes, deux conditions doivent être observées : toutes les actions doivent s'exécuter et elles doivent le faire dans le bon ordre. Pour que l'attribut "Etat" évolue correctement, les axiomes suivants sont nécessaires :

a. Initialiser $\rightarrow$ XEtat = 0
b. Changement$_1$ $\vee$ ... $\vee$ Changement$_m$ $\rightarrow$ XEtat = Etat + 1

Avec $1 \leq i \leq m$

Au début de la séquence, "Etat" est initialisé à zéro et chaque fois qu'une action est exécutée, nous avançons d'un pas, donc, "Etat" est augmenté de un.

Si la séquence comprend les m actions différentes "Action$_1$", ..., "Action$_m$", elle se déroulera dans le bon ordre à condition que les axiomes nous disent que $\forall\ 1 \leq i \leq m$, "Action$_i$" ne s'exécute que si nous nous trouvons dans un état tel qu'elle doit s'exécuter. De même, $\forall\ 1 \leq i \leq m$, si nous nous trouvons dans un état tel qu'une action doit s'exécuter, "Action$_i$" se déroulera tôt ou tard.

Comment déterminer l'état dans lequel une action doit s'exécuter? Une action "Action$_i$" quelconque avec $1 \leq i \leq m$ peut se réaliser à différents endroits dans la séquence. Si elle doit s'exécuter aux positions i_1, ..., i_p (i_1, ..., i_p distincts et compris entre un et n), l'état dans lequel "Action$_i$" devra s'exécuter sera "(Etat = i_1 - 1) $\vee$ ... $\vee$ (Etat = i_p - 1)" Nous aurons donc les axiomes suivants :

c. (Etat = i_1 - 1) $\vee$ (Etat = i_2 - 1) $\vee$ ... $\vee$ (Etat = i_p - 1) $\rightarrow$ FChangement$_i$
d. Changement$_i$ $\rightarrow$ (Etat = i_1 - 1) $\vee$ (Etat = i_2 - 1) $\vee$ ... $\vee$ (Etat = i_p - 1)

Avec $1 \leq i \leq m$

Pour nous assurer que l'objet "ETAT" soit complet, nous devons imposer certaines conditions aux ensembles $\{i_1, ..., i_p\}$:
1. i_1, ..., i_p sont distincts et $p \geq 1$.
2. la même action doit s'exécuter aux positions i_1, ..., i_p.
3. $\forall\ j \notin \{i_1, ..., i_p\}$ $(1 \leq j \leq n)$, les actions devant s'exécuter aux positions i_1 et j sont différentes.

De plus, l'union des ensembles $\{i_1, ..., i_p\}$ avec $1 \leq i \leq m$ $(m \leq n)$ est égale à $\{1, ..., n\}$.

Afin de garantir le respect de ces conditions, nous définissons ci-après une manière de construire ces ensembles.

Au départ de cette construction, l'ensemble initial sera l'ensemble des naturels compris entre 1 et n, à savoir {1, ..., n}. Nous l'appellerons "ensembleinitial".

Etape 1

Le premier ensemble, "ensemble$_1$", est l'ensemble vide.

a. ensemble$_1$ = ensemble$_1$ + {1}
   ensemblerestant = ensembleinitial / {1}

b. Nous prenons le premier j appartenant à "ensemblerestant" tel que l'action devant s'exécuter à la position j est la même que celle devant s'exécuter à la position 1.
   ensemble$_1$ = ensemble$_1$ + {j}
   ensemblerestant = ensemblerestant / {j}

Nous répétons l'étape b. tant qu'il existe un j répondant aux conditions.

"ensemble$_1$" répond bien aux conditions.

1. Oui car il existe au moins un élément dans "ensemble$_1$" et 1_1, ..., 1_p sont distincts puisque à l'étape b., nous prenions j dans "ensemblerestant" et les éléments qui étaient déjà dans "ensemble$_1$" ont été ôtés de "ensemblerestant".
2. Oui car dans l'étape b., nous prenions j d'après ce critère.
3. Oui car nous avons pris tous les j qui correspondent aux positions de la même action.

Etape i

Nous avons déjà construit i - 1 ensembles qui répondent aux conditions. Il nous reste un ensemble de naturels compris entre 1 et n qui n'appartiennent pas à l'union des ensembles déjà construits. Ceux-ci n'ont pas encore été ôtés à "ensembleinitial" et se retrouvent donc dans "ensemblerestant". Si celui-ci est vide, la construction est achevée. S'il n'est pas vide, nous construisons "ensemble$_i$".

Au début de l'étape, "ensemble$_i$" est vide.

a. Nous prenons le premier élément de "ensemblerestant". Ce sera l'élément {i_1}.
   ensemble$_i$ = ensemble$_i$ + {i_1}
   ensemblerestant = ensembleinitial / {i_1}

b. Nous prenons le premier j appartenant à "ensemblerestant" tel que l'action devant s'exécuter à la position j est la même que celle devant s'exécuter à la position i_1.
   ensemble$_i$ = ensemble$_i$ + {j}

ensemblerestant = ensemblerestant / {j}

Nous répétons l'étape b. tant qu'il existe un j répondant aux conditions.

"ensemble$_i$" répond bien aux conditions.

1. Oui car "ensemble$_i$" comprend au moins l'élément $\{i_1\}$. i_1, ..., i_p sont distincts puisque à l'étape b., nous prenions j dans "ensemblerestant" et les éléments qui étaient déjà dans "ensemble$_i$" avaient été ôtés de "ensemblerestant".
2. Oui car dans l'étape b., nous prenions j d'après ce critère.
3. C'est vrai pour les éléments de "ensemblerestant" car nous avons pris tous les j qui correspondent aux positions de la même action. Nous savons que les i - 1 premiers ensembles répondent aux conditions. Les éléments de "ensemblerestant" du début de l'étape étaient donc tels que les actions correspondant à ces positions étaient différentes des actions correspondant aux positions reprises dans les i - 1 premiers ensembles.

Après un nombre fini m d'étapes ($m \leq n$), "ensemblerestant" sera vide. Nous pourrons alors construire les axiomes du groupe c. et d.

6.2.1.2. Construction de la structure de conception

L'objet "ETAT" étant créé, construisons la structure de conception de manière à synchroniser ses actions avec celles du noyau et de l'interface de l'action "A".

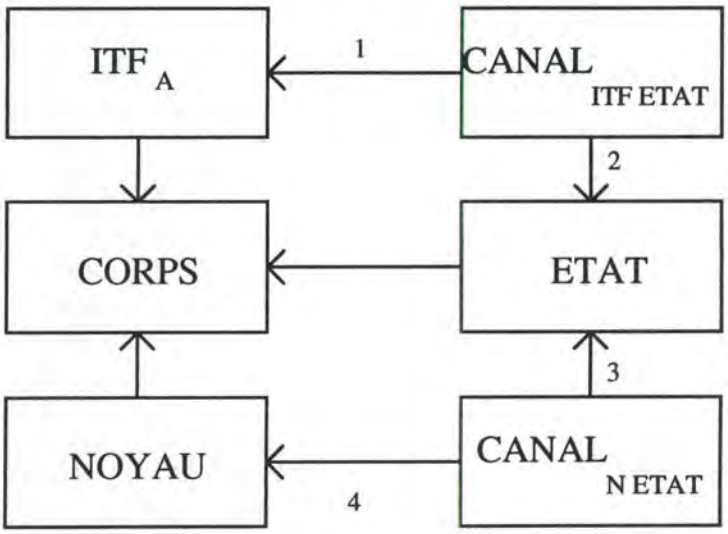

Fig. 6.1 : *Schéma général de la structure de conception pour une séquence.*

Où l'objet ITF_A est construit tel que défini dans la théorie.

La description des canaux est :

CANAL ITF ETAT

Sorts : /
Symboles d'opération : /
Symboles d'attribut : /
Symboles d'action : Act_0

CANAL N ETAT

Sorts : /
Symboles d'opération : /
Symboles d'attribut : /
Symboles d'action : Act_1
...
Act_m

Les différents composants de cette structure sont reliés par les morphismes 1, 2, 3 et 4. Leur descritption est la suivante :

1

Relation : CANAL $_{ITF\ ETAT} \rightarrow ITF_A$
sorts : /
Symboles d'action : $Act_0 \rightarrow Beg_A$

2

Relation : CANAL $_{ITF\ ETAT} \rightarrow ETAT$
sorts : /
Symboles d'action : $Act_0 \rightarrow$ Initialiser

3

Relation : CANAL $_{N\ ETAT} \rightarrow$ ETAT
sorts : /
Symboles d'action : $Act_1 \rightarrow Changement_1$
...
$Act_m \rightarrow Changement_m$

4

Relation : CANAL $_{N\ ETAT} \rightarrow N$
sorts : /
Symboles d'action : $Act_1 \rightarrow Action_1$
...
$Act_m \rightarrow Action_m$

Pour que le corps soit complet, nous devons préciser à quel moment l'action "A" se terminera. Le corps sera donc la colimite de la structure de conception de la figure 6.1. à laquelle nous ajouterons les axiomes suivants :

e. $\mathbf{In_A \wedge (Etat = n) \rightarrow FEnd_A}$
f. $\mathbf{End_A \rightarrow In_A \wedge (Etat = n)}$

Remarque : il peut se produire qu'une action à implémenter possède déjà la granularité de changement de niveau inférieur. Nous nous trouverons alors dans le cas limite où la séquence comporte une seule action devant être exécutée une fois. Dans ce cas de figure, nous construirons le corps selon la méthode ci-dessus.

6.2.2. Représentation d'une boucle

La boucle que nous avons choisie de représenter dans notre logique sera une boucle "tant que", c'est à dire que la condition de fin sera testée avant l'entrée dans la boucle. Nous

considérons que la boucle porte sur une action; celle-ci pourra, par des raffinements ultérieurs, acquérir une structure plus complexe.

Dans ce point, nous allons voir comment implémenter deux types de boucles. Le cas le plus simple, c'est à dire celui où la condition de fin de boucle porte sur des attributs de l'objet, est présenté en premier lieu et est suivi de la méthode d'implémentation d'une boucle lorsque celle-ci nécessite un compteur.

6.2.2.1. Boucle dont la condition de fin porte sur les attributs de l'objet

Si nous avons une action à exécuter tant qu'une certaine condition est réalisée, le principe de la boucle en elle-même est assez simple. Il nous suffira d'introduire les axiomes suivants dans le corps :

a. $In_A \wedge condition \rightarrow FAction$
b. $In_A \wedge \neg condition \rightarrow FEnd_A$
c. $End_A \rightarrow In_A \wedge \neg condition$

où condition est une formule du premier ordre sur les attributs du noyau.

L'axiome a. nous assure que tant que l'action abstraite "A" s'exécute et que la condition est réalisée, l'action à exécuter se déroulera tôt ou tard. Les axiomes b. et c. gèrent la fin de l'action "A". Si la condition n'est plus réalisée, "A" se terminera tôt ou tard et elle ne pourra se terminer que dans ce cas.

Il faudra évidemment tenir compte du fait que des actions qui se déroulent en parallèle peuvent interférer avec l'évaluation de la condition. Nous aborderons le problème de la gestion de la concurrence entre les actions dans le point 6.3.

6.2.2.2. Boucle qui nécessite un compteur

Un problème se pose si un compteur est nécessaire pour réaliser la boucle telle que vue dans l'exemple du chapitre 4. Les attributs du noyau ne comprenant pas de compteur, nous devons en ajouter un. Dans notre optique orientée objet, nous construisons un objet dont le but est de gérer ce compteur.

Construction de l'objet "RESTE"

L'objet "RESTE" doit comprendre un attribut "Reste" qui peut être modifié par les actions "Initialiser" et "Décroître". L'attribut "Reste" représente le nombre d'itérations restant à exécuter.

L'objet comprend les axiomes suivants :

a. Initialiser(x) → XReste = x
b. Décroître → XReste = prec(Reste)
c. Décroître → (Reste > 0) = true
d. (Reste > 0) = true → FDécroître

Il apparaît que le reste peut être initialisé et que x itérations seront encore à exécuter. L'action "Décroître" a pour effet de diminuer le reste de un. Cette action ne peut se faire que si le reste est supérieur à zéro, c'est à dire s'il reste au moins une itération à exécuter. De plus, tant que le reste est supérieur à zéro, tôt ou tard, l'action "Décroître" se réalisera.

Construction de la structure de conception

Nous devons maintenant synchroniser les actions de l'objet "RESTE" avec les actions du noyau et de l'interface de "A". Nous synchronisons l'action "Initialiser" avec l'action "Beg_A". Cette action doit avoir un naturel comme paramètre. L'action "Décroître" est synchronisée avec l'action à exécuter à l'intérieur de la boucle. Cela nous donne la structure de conception suivante :

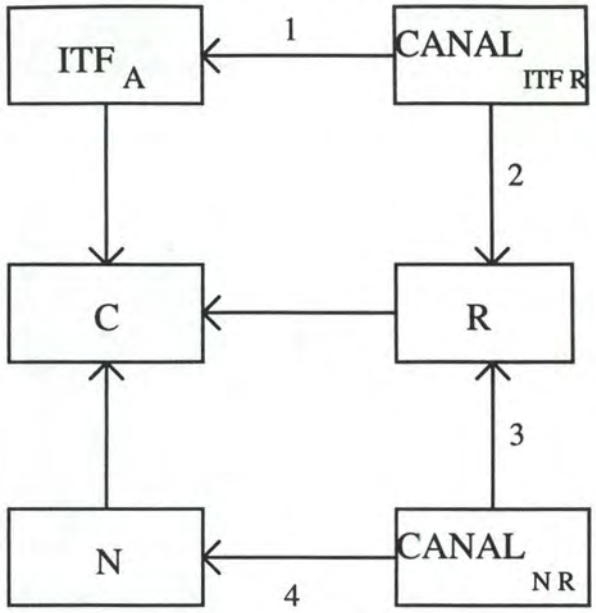

Fig 6.2. : *Schéma général de la structure de conception pour une boucle.*

Où l'objet ITF_A est construit tel que défini dans la théorie.

La description des canaux est :

CANAL ITF R

Sorts : /
Symboles d'opération : /
Symboles d'attribut : /
Symboles d'action : Init

CANAL N R

Sorts : /
Symboles d'opération : /
Symboles d'attribut : /
Symboles d'action : Dec

Les différents composants de cette structure sont reliés par les morphismes 1, 2, 3 et 4. Leur description est la suivante :

1

Relation : CANAL $_{ITF\ R} \rightarrow ITF_A$
sorts : /
Symboles d'action : Init $\rightarrow Beg_A$

2

Relation : CANAL $_{ITF\ R} \rightarrow R$
sorts : /
Symboles d'action : Init $\rightarrow$ Initialiser

3

Relation : CANAL $_{N\ R} \rightarrow R$
sorts : /
Symboles d'action : Dec $\rightarrow$ Décroître

4

Relation : CANAL $_{N\ R} \rightarrow N$
sorts : /
Symboles d'action : Dec $\rightarrow$ Action

Pour que le corps ait le comportement désiré, il faut préciser les conditions pour lesquelles l'action abstraite "A" peut se terminer, c'est à dire pour que l'action "End_A" puisse s'exécuter.

Nous allons donc ajouter à la colimite de la structure de conception présentée ci-dessus les axiomes suivants :

e. $\mathbf{In_A \wedge (Reste = 0) \rightarrow FEnd_A}$
f. $\mathbf{End_A \rightarrow In_A \wedge (Reste = 0)}$

6.2.3. Représentation du "if ... then ... else ..."

Représentons la structure "if condition then Action1 else Action2" où la condition est une formule du premier ordre sur les attributs de l'objet abstrait. Les actions "Action1" et "Action2" peuvent grâce à des raffinements successifs représenter des structures plus complexes.

Pour implémenter une action sur cette structure, il faut que le "if ... then ... else ..." se réalise une et une seule fois. Pour savoir s'il a déjà été exécuté, nous pourrions introduire un nouvel attribut dans le noyau mais il est préférable de construire un objet qui gérera cet attribut.

6.2.3.1. Construction de l'objet "IF"

Cet objet comporte l'attribut "Ifréalisé", de type booléen, les trois actions "Initialiser", "Act1" et "Act2" ainsi que les axiomes suivants :

a. Initialiser → Ifréalisé = false
b. Act1 → XIfréalisé = true
c. Act2 → XIfréalisé = true

6.2.3.2. Construction de la structure de conception

Synchronisons l'action "Initialiser" avec l'action "Beg_A" et les actions "Act1" et "Act2" respectivement avec "Action1" et "Action2". Construisons la structure de conception de la manière suivante :

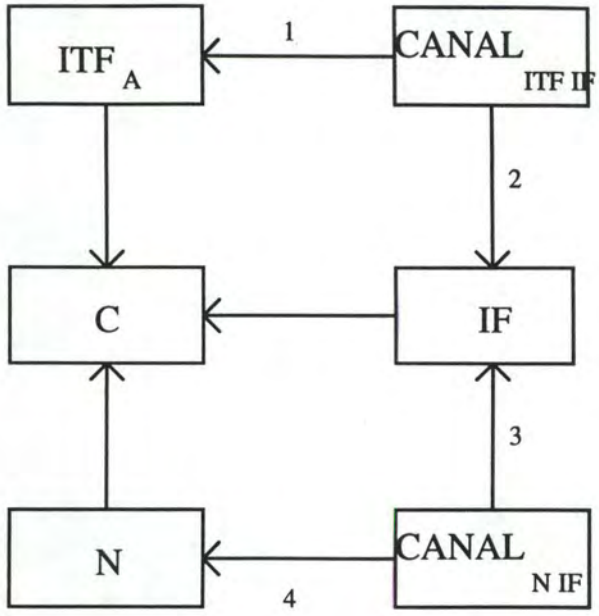

Fig 6.3. : *Schéma général de la structure de conception pour une boucle.*

Où l'objet ITF_A est construit comme nous l'avons défini dans la théorie.

La description des canaux sera :

CANAL ITF IF

Sorts : /
Symboles d'opération : /
Symboles d'attribut : /
Symboles d'action : Init

CANAL N IF

Sorts : /
Symboles d'opération : /
Symboles d'attribut : /
Symboles d'action : A1, A2

Les différents composants de cette structure sont reliés par les morphismes 1, 2, 3 et 4. Leur description est la suivante :

1

Relation : CANAL $_{ITF\ IF} \rightarrow ITF_A$
sorts : /
Symboles d'action : Init $\rightarrow Beg_A$

2

Relation : CANAL $_{ITF\ IF} \rightarrow$ IF
sorts : /
Symboles d'action : Init $\rightarrow$ Initialiser

3

Relation : CANAL $_{N\ IF} \rightarrow$ IF
sorts : /
Symboles d'action : A1 $\rightarrow$ Act1
A2 $\rightarrow$ Act2

4

Relation : CANAL $_{N\ IF} \rightarrow$ N
sorts : /
Symboles d'action : A1 $\rightarrow$ Action1
A2 $\rightarrow$ Action2

Pour que le corps soit complet et qu'il ait le comportement désiré, nous devons assurer que le "if ... then ... else ..." se réalisera une et une seule fois. Nous ajouterons au corps les axiomes suivants :

d. $\mathbf{In_A \wedge \neg Ifréalisé \wedge condition \rightarrow FAction1}$
e. $\mathbf{In_A \wedge \neg Ifréalisé \wedge \neg condition \rightarrow FAction2}$
f. $\mathbf{Action1 \rightarrow In_A \wedge \neg Ifréalisé \wedge condition}$
g. $\mathbf{Action2 \rightarrow In_A \wedge \neg Ifréalisé \wedge \neg condition}$
h. $\mathbf{End_A \rightarrow In_A \wedge Ifréalisé}$
i. $\mathbf{In_A \wedge Ifréalisé \rightarrow FEnd_A}$

6.3. Gestion de la concurrence entre les actions

Dans le point 6.2. nous avons construit une méthode générale d'implémentation d'une action en termes d'un objet concret. Celle-ci ne tenait pas compte du fait que nous nous trouvions dans un système concurrent et donc que plusieures actions pouvaient s'exécuter simultanément. Ici toutefois, nous analysons cette concurrence afin de dégager les problèmes éventuels et leur apporter une solution. Cela conduira à obtenir une méthode d'implémentation générale d'un objet abstrait en termes d'un objet plus concret.

Dans un premier temps, voyons quels sont les problèmes liés à la concurrence à l'intérieur d'un objet. Ensuite, envisageons ce que ces problèmes vont impliquer au niveau de l'implémentation de l'objet abstrait et les solutions à apporter.

6.3.1. Concurrence à l'intérieur d'un objet

Nous avons défini pour les objets, les ensembles $\Theta^\bullet$ et $\Xi^\bullet$. Si les actions "Action1" et "Action2" s'exécutent au même instant, des problèmes peuvent se poser. En effet, ces actions peuvent agir sur les mêmes attributs et donc interférer entre elles.

Pour voir si les actions interfèrent entre elles, analysons l'intersection entre les ensembles $\Theta^\bullet$(Action1), $\Xi^\bullet$(Action1), $\Theta^\bullet$(Action2) et $\Xi^\bullet$(Action2). Plusieurs cas peuvent se présenter :

1. $\{(\Theta^{\bullet}(\text{Action1}) \cup \Xi^{\bullet}(\text{Action1})) \cap (\Theta^{\bullet}(\text{Action2}) \cup \Xi^{\bullet}(\text{Action2}))\} = \varnothing$

Ce cas est le plus simple; il ne pose aucun problème de concurrence car les deux actions travaillent sur des attributs différents.

2. $\{\Xi^{\bullet}(\text{Action1}) \cap \Xi^{\bullet}(\text{Action2})\} \neq \varnothing$

Si deux actions modifient le même attribut, un problème immédiat se présentera au niveau de l'objet. En effet, les occurences de "Action1" et de "Action2" donnent une certaine valeur à cet attribut. Si celles-ci surviennent au même instant et étant donné l'atomicité des actions, l'attribut devra prendre deux valeurs différentes à l'instant suivant. Au niveau de l'objet, ce problème sera résolu par les axiomes. Nous aurons :

Action1 $\rightarrow$ **X**att = valeur1

Action2 $\rightarrow$ **X**att = valeur2

Si "valeur1" et "valeur2" sont différents, nous ne pourrons pas avoir en même temps "**X**att = valeur1" et "**X**att = valeur2". D'après les règles du calcul propositionnel, "Action1" et "Action2" ne peuvent s'exécuter au même instant.

3. $\{\Theta^{\bullet}(\text{Action1}) \cap \Xi^{\bullet}(\text{Action2})\} \neq \varnothing$ OU $\{\Xi^{\bullet}(\text{Action1}) \cap \Theta^{\bullet}(\text{Action2})\} \neq \varnothing$

Nous nous trouvons dans le cas où une des deux actions modifie un attribut que l'autre action lit. Or, selon la définition de l'atomicité, si une action modifie un attribut, celui-ci prendra sa nouvelle valeur à l'instant suivant. Ce cas ne posera donc aucun problème au niveau de l'objet.

4. $\{\Theta^{\bullet}(\text{Action1}) \cap \Theta^{\bullet}(\text{Action2})\} \neq \varnothing$

Le cas où les deux actions lisent le même attribut ne pose aucun problème.

Remarque : nous dirons que deux actions entrent en conflit au niveau de leur attributs si nous nous trouvons dans le deuxième ou le troisième cas.

6.3.2. Implémentation de deux actions concurrentes

Lors de l'implémentation de plusieurs actions en termes d'un même objet concret, nous pouvons rencontrer plusieurs types de problèmes. Ceux qui proviennent du fait que deux actions abstraites ne peuvent s'exécuter en même temps et ceux dûs aux conflits d'attributs au niveau concret.

6.3.2.1. Implémentation de deux actions qui entrent en conflit d'écriture au niveau de leurs attributs

Lorsque ces actions sont atomiques, autrement dit, lorsque nous nous trouvons au niveau de l'objet abstrait, nous avons vu dans le point 6.3.1. que ces problèmes étaient résolus automatiquement. Pour implémenter ces actions en termes d'un objet plus concret, il faut spécifier dans l'objet concret que ces deux actions ne peuvent survenir dans le même intervalle de temps. Pour décrire cela, nous ajouterons dans le corps des axiomes qui empêchent ces actions de s'exécuter à l'intérieur du même intervalle. Si les deux actions à implémenter sont "A" et "B", telles qu'au niveau abstrait nous avons :

$A \rightarrow \mathbf{X}att = valeur1$

$B \rightarrow \mathbf{X}att = valeur2$;

au niveau concret nous aurons les axiomes :

$\mathbf{Beg_A} \rightarrow \neg(\mathbf{In_B} \vee \mathbf{Beg_B})$

$\mathbf{Beg_B} \rightarrow \neg(\mathbf{In_A} \vee \mathbf{Beg_A})$

qui empecheront "A" et "B" de se dérouler dans un même intervalle.

6.3.2.2. Implémentation de deux actions qui entrent en conflit au niveau concret

L'implémentation de deux actions donne lieu à un conflit au niveau concret si lors de leur exécution, elles agissent sur les mêmes attributs concret. Ce problème peut être causé par deux choses :

1°) il existe un conflit de lecture-ecriture au niveau de l'objet abstrait.

2°) les actions concrètes qui implémentent une action abstraite entrent en conflit avec les actions concrètes en implémentant une autre.

Dans les point A. et B., nous analyser ces deux cas et donner une solution théorique au problèmes qu'ils engendrent. Dans le point C., nous verrons comment les réaliser de manière concrète.

A. Conflit de lecture-écriture au niveau de l'objet abstrait

Prenons maintenant le cas de deux actions abstraites "A" et "B".

A → **X**att1 = att2

B → **X**att2 = valeur

Nous avons vu au point 6.3.1. qu'il n'y avait pas d'exclusion mutuelle de ces deux actions au niveau abstrait, elles pourront dès lors se dérouler au même instant.

Supposons que

- "A" soit réalisée à l'aide d'une action "c" qui lit "att2".
- "B" soit réalisée à l'aide d'une action "d" qui modifie "att2".

Un déroulements possibles nous est donné par la figure 6.4.

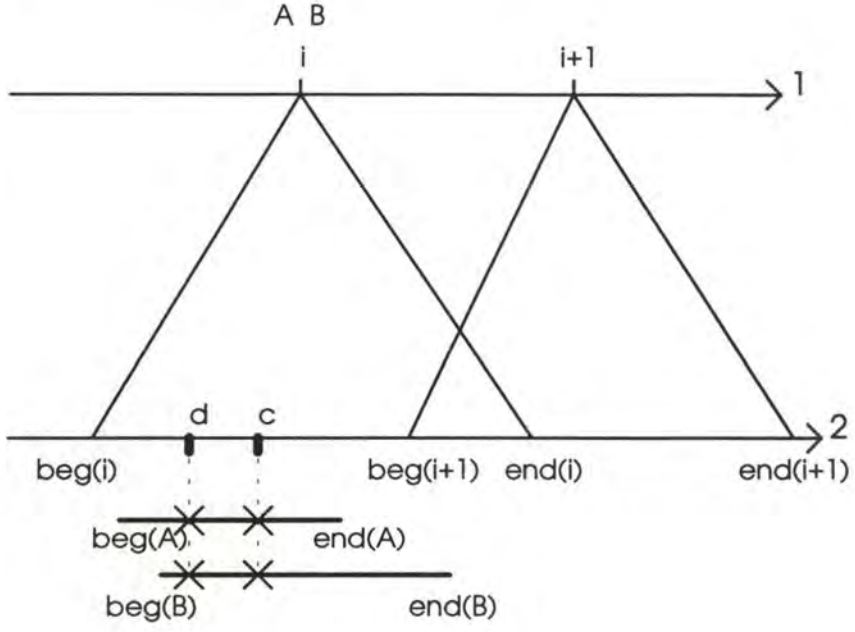

Fig. 6.4. : *Exemple de déroulement incorrect de deux actions abstraites "A" et "B" qui entrent en conflit de lecture-ecriture.*

Cette interprétation est telle que l'action "d" modifie la valeur de l'attribut "att2" avant que l'action "c" ne le lise. Dès lors "c" lit l'attribut "att2" dont la valeur n'est plus celle du début de l'intervalle. Un tel déroulement de l'action abstraite "A" donne un résultat erroné par rapport à ce qui est attendu.

Pour que l'exécution des actions "A" et "B" soit correcte, nous créons un tampon propre à chacune d'entre elles. Autrement dit, les actions qui implémentent une action abstraite agissent uniquement sur le tampon lié à celle-ci. Celui-ci est mis à jour de la manière suivante :
-si l'exécution d'une action abstraite "Act" nécessite la lecture d'un attribut abstrait "att", au début de l'exécution, nous mettons la valeur de "att" dans le tampon.
-si l'exécution d'une action abstraite "Act" nécessite de modifier un attribut abstrait "att", à la fin de l'exécution, nous mettons la valeur du tampon dans "att".

Dans notre exemple, nous créons deux tampons "$att2_A$" et "$att2_B$" respectivement liés à "A" et à "B". Au début de l'exécution de "A", nous mettons la valeur de "att2" dans "$att2_A$" et à la fin de l'exécution de "B", nous mettons la valeur de "$att2_B$" dans "att2".

B. Conflit au niveau des attributs de l'objet concret

Prenons deux actions abstraites "A" et "B" telles que l'implémentation de "A" nécessite une action concrète "c" et celle de "B" nécessite une action concrète "d". "c" et "d" peuvent entrer en conflit, par exemple si "d" modifie des attributs lus par "c".
Nous avons par exemple "c" et "d" au niveau concret, telles que :
c → **X**att1 = attconcret ;
d → **X**attconcret = valeur.

Si les actions "A" et "B" se déroulent comme nous l'avons illustré à la figure 6.4., l'exécution de "A" est faussée dans le cas ou "attconcret" est local à l'exécution de chacune des actions. Pour résoudre ce problème, l'action "A" a ses attributs propres, ils ne peuvent être modifiés que par l'occurrence d'une action implémentant "A". Etant donné qu'ils sont locaux à "A", il n'y a pas besoin de faire de mise à jour.

C. Réalisation concrète des solutions apportées aux conflits

La solution que nous avons choisie pour résoudre les problèmes apparus aux points A et B est de créer des attributs qui seront locaux à l'action à implémenter. Pour réaliser cela, nous avons pour chaque action à implémenter un noyau qui sera propre à la construction de sa structure de conception.

Afin de ne pas dupliquer des actions et des attributs inutiles, nous allons construire un noyau minimal pour l'implémentation de "A", "$Noyau_A$", de la manière suivante :
Soit {Act1, ..., Actn} l'ensemble des actions qui implémentent "A". Nous construisons "$Noyau_A$" par élimination sur le noyau initial.

1°) Nous prenons le noyau et nous ôtons toutes les actions n'appartenant pas à l'implémentation de "A" : $\Gamma_{Noyau_A}{}^{\bullet} = \{Act1, ...,Actn\}$.

2°) Nous enlevons tous les attributs qui ne sont pas modifiés ou lus par "Acti" ($1 \leq i \leq n$) : Si nous définissons pour "$Noyau_A$", les ensembles Λ et M tels que définis au point 6.3.3., nous avons :

$\forall att \in A_{Noyau_A}{}^{\bullet}, \exists act \in \Gamma_{Noyau_A}{}^{\bullet}$ tel que $act \in \Theta_{Noyau_A}(att)$ ou $act \in \Xi_{Noyau_A}(att)$.

$\neg\exists(att \notin A_{Noyau_A}{}^{\bullet}$ et $act \in \Gamma_{Noyau_A}{}^{\bullet})$ tel que $act \in \Theta_{Noyau_A}(att)$ ou $act \in \Xi_{Noyau_A}(att)$.

3°) Nous supprimons tous les axiomes qui ne portent pas sur un attribut ou une action de l'objet construit d'après les points 1° et 2°.

Pour chaque action "A" à implémenter, nous créons la structure de conception telle que définie au point 6.2. sur "$Noyau_A$".

Schématiquement, cela nous donnera :

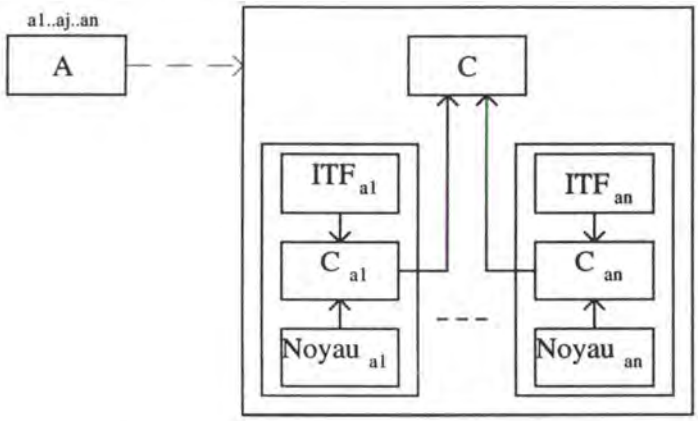

FIG 6.5. : *Structure de conception générale pour l'implémentation de l'objet abstrait "A"*

Avec cette méthode de construction du corps, chaque implémentation d'une action possède ses actions et ses attributs propres. Si ce sont des attributs locaux à l'exécution d'une action, cela ne pose pas de problèmes.

Dans le cas où l'attribut concret est la traduction d'un attribut abstrait, celui-ci doit être partagé par toutes les actions qui peuvent agir dessus. Nous ajoutons donc dans le corps un attribut qui fait le lien entre les différentes images propres à l'exécution de chaque action. C'est sur cet attribut que se fera la démonstration des propriétés. Cela est réalisé de la manière

suivante:

supposons une action "A" qui agit sur l'attribut abstrait "Att" et "AttC" qui est sa traduction au niveau concret (AttC = ι(Att)). "Noyau$_A$" possède un attribut "AttC$_A$" qui sera l'image de "AttC" lors de l'exécution de "A".

Nous construisons la structure de conception telle qu'illustrée à la figure 6.5. et nous ajoutons dans "C", l'attribut "AttC". Comme "AttC" fait le lien entre les différentes images des actions, nous représentons le lien entre l'attribut et son image par les axiomes :

Beg$_A$ → XAttC$_A$ = AttC

End$_A$ → XAttC = AttC$_A$

Or, d'après la condition de localité de "Noyau$_A$", "AttC$_A$" ne peut être modifié que par l'occurrence d'une de ses actions. Nous devons donc ajouter une action "Initialiser" à l'intérieur de "Noyau$_A$". C'est la quatrième étape de la construction de "Noyau$_A$".

4°) Si "A" agit sur des attributs abstraits, nous ajoutons "Initialiser".

Cette action est synchronisée avec "Beg$_A$" à l'aide d'un canal. Il est relié aux objets "ITF$_A$" et "Noyau$_A$" par des morphismes.

6.4. Conclusion

Maintenant que nous avons vu comment implémenter une action abstraite en termes d'objets concrets et comment gérer la concurrence entre plusieurs actions, nous sommes à même de donner un algorithme général d'implémentation d'un objet abstrait.

Prenons le cas général d'un objet abstrait qui contient les actions {Act1, ..., Actn} et les attributs {Att1, ..., Attm}. Nous construirons l'implémentation de la manière suivante :

1°) ∀ Acti ($1 \leq i \leq n$), construire un noyau minimal (point 6.3.2.2.C.).

2°) ∀ Acti ($1 \leq i \leq n$), construire la structure de conception sur "Noyau$_{Acti}$" (point 6.2.)

3°) Si Acti agit sur un attribut abstrait, construire un canal pour synchroniser l'action "Initialiser" de "Noyau$_{Acti}$" et "Beg$_{Acti}$" (point 6.3.2.2.C.).

4°) La structure générale étant telle qu'illustrée à la figure 6.4., "C" est la somme des différents "C$_{Acti}$".

5°) Ajouter à "C" les attributs concrets qui sont la traduction des attributs abstraits. Ces attributs sont nommés "AttC$_1$", ..., "AttC$_p$" (point 6.3.2.2.C.).

6°) ∀ Acti ($1 \leq i \leq n$) et ∀ AttC$_j$ ($1 \leq j \leq p$), si l'implémentation de "Acti" lit "AttC$_j$", ajouter au corps l'axiome :

Beg$_{Acti}$ → XAttC$_{1Acti}$ = AttC$_1$

Si l'implémentation de "Acti" modifie "AttC$_j$", ajouter au corps l'axiome :

$\mathbf{End_{Acti} \rightarrow XAttC_1 = AttC1_{Acti}}$

(point 6.3.2.2.C.)

7°) $\forall$ Acti ($1 \leq i \leq n$) et $\forall$ Actj ($1 \leq j \leq n$) ($i \neq j$), si "Acti" entre en conflit avec "actj", ajouter les axiomes :

$\mathbf{Beg_{Acti} \rightarrow \neg(In_{Actj} \vee Beg_{Actj})}$

$\mathbf{Beg_{Actj} \rightarrow \neg(In_{Acti} \vee Beg_{Acti})}$

(point 6.3.2.1.)

8°) Mettre dans le corps :

a) les axiomes qui expriment la condition de localité de l'objet abstrait.

b) les axiomes qui expriment la condition de Sécurité, de vivacité et de début de l'objet abstrait.

(point 4.5.1.1.)

Chapitre 7

Conclusion

Tout au long de ce mémoire, nous nous sommes appliqués à développer des exemples pour montrer la cohérence de la théorie développée par J-L. Fiadeiro et T. Maibaum et de donner des systèmes de preuve pour démontrer que l'objet abstrait conserve ses propriétés lorsqu'il est implémenté au niveau concret.

Après avoir résumé de la théorie, nous l'avons appliquée à plusieurs exemples de manière intuitive. Ceux-ci nous ont permis de nous rendre compte que certaines parties de la théorie devaient être corrigées ou pouvaient être améliorées. Les changements apportés et développés aux chapitres 4 et 5 sont :

1°) l'ajout de deux nouveaux types d'axiomes (point 3.2.2.).

2°) la modification de la règle de changement (point 4.5.1.2.).

3°) la suppression de deux axiomes inutiles dans l'interface (point 5.2.).

4°) l'ajout de la notion d'attribut lu ou écrit par une action et toutes les modifications théoriques que cela implique (point 5.3.)

En développant les exemples, nous avons apporté des solutions ponctuelles aux problèmes qui sont apparus. Dans le dernier chapitre, nous établissons un système général d'implémentation d'un objet abstrait en termes d'objets plus concrets. Celui-ci reprend toutes les solutions apportées aux différents problèmes pouvant être rencontrés au cours de l'implémentation d'un objet abstrait. Il permet d'implémenter une action en termes d'objets concrets et gère la concurrence au niveau concret.

Ce mémoire, n'a pas développé le problème de l'implémentation des types de données. Il serait intéressant de réaliser une analyse complète combinant le rafinnement des actions ainsi que celui des types de données.

Bibliographie

[Aceto & Hennessy 89]
L. Aceto et M. Hennessy, Towards Action Refinement in Process Algebras", in *Proc. 4th* LICS, IEEE 1989, 138-145.

[Ehrig & Mahr 85]
H.Ehrig et B. Mahr ,"*Fundamentals of Algebraic Specification 1*", W.Brauer, G.Rozenbrg, A.Salomaa (Eds.), Springer-Verlag 1985.

[Ehrig & Mahr 90]
H.Ehrig et B. Mahr ,"*Fundamentals of Algebraic Specification 2*", W.Brauer, G.Rozenbrg, A.Salomaa (Eds.), Springer-Verlag1990.

[Fiadeiro & Maibaum 92]
J.L. Fiadeiro et T. Maibaum, "Temporal Théories as Modularisation Units for Concurrent System Specification", *Formal Aspects of Computing* 4(3), 1992, 239-272.

[Fiadeiro & Maibaum 93A]
J.L. Fiadeiro et T. Maibaum, "Design Structures for Reactive Systemes", research notes, University of Lisbon, Portugal, 1993.

[Fiadeiro & Maibaum 93B]
J.L. Fiadeiro et T. Maibaum, "Actions are Objects : Refinement in the Temporal Logic of Objects", *Proc. 3rd ISCORE Workshop*, U.Lipeck and G.Koschorreck (eds), Hannover 1992, 239-272.

[Goldblatt 92]
Robert Goldblatt, "*Logics of time and computation*", Second Edition (Revised and Expanded), CSLI 1992.

[Lecharlier 91]
Cours de "*Preuve de programme*", 1991, donné par B. Lecharlier.

[Thayse 89]
A.Tayse & co-auteurs, "*Approche logique de l'intelligence artificielle. 2. De la logique modale à la logique des bases de données*", Dunod, 1982.

[Veloso & al 85]
P. Veloso, T. Maibaum et M. Salder, "Program Development and theory Manipulation", *Proc. Third International Workshop on Software Specification and Design*, London, IEEE Computer Society Press 1985, 228-232.

Table des matières

Chapitre 1 3
Introduction 3
1.1 Pré-requis 5
1.2 Organisation du texte 5
Chapitre 2 6
Concepts théoriques 6
2.1. Introduction 6
2.2. Les spécifications algébriques 6
2.3. La logique temporelle linéaire 8
2.3.1. Introduction 8
2.3.2. Rappel de la notation BNF 9
2.3.3. Logique modale 9
2.3.3.1. Formules modales 9
2.3.3.2. Frames et modèles 11
2.3.3.3. Vérité et validité 12
2.3.4. La logique temporelle proprement dite 12
2.3.4.1. Symboles 12
2.3.7. Axiomatisation 13
Chapitre 3 15
Présentation de la théorie initiale 15
3.1. Introduction 15
3.2. Description de composants individuels 15
3.2.1. Signature d'objet. 16
3.2.2. Description des objets 18
3.3 Description d'un système 22
3.3.1 La notion de morphisme 22
3.3.2 Création de liens entre descriptions d'objets (Canaux) 23
3.3.3 Sommer deux objets (Colimite) 25
3.4 Démonstration de propriétés 29
3.4.1 Système de démonstration 29
3.4.2 Démonstration d'un invariant 30
3.5 Implémentation d'objets abstraits 32
3.5.1 Différence de granularité 33
3.5.2 Méthode d'implémentation 35
3.5.3 Gestion de la concurrence 36
Chapitre 4 40
Développement d'un exemple 40
4.1. Introduction 40
4.2. Description de l'objet abstrait 40
4.3. Description de l'objet concret 41
4.4. Raffinement des actions sur des objets 42
4.5. Construction du Système 45

4.5.1. Première approche 45
4.5.1.1. Construction du corps 45
4.5.1.2. Démonstration des propriétés 50
4.5.2. Seconde approche 63
4.5.2.1. Construction du corps 64
4.5.2.2. Démonstration des propriétés 70
Chapitre 5 73
Modifications de la théorie 73
5.1. Introduction 73
5.2. Modification des axiomes des interfaces 73
5.3. Optimisation de la gestion de la concurrence et répercussions sur la théorie 75
5.3.1. Extension de la signature d'objet 76
5.3.2. Raffinement de la condition de localité 76
5.3.3. Conditions sur l'abstraction 85
5.4. Développement d'un exemple 87
5.4.1. Description de l'objet abstrait. 87
5.4.2. Description de l'objet concret 87
5.4.3. Construction du corps 88
5.4.4. Preuve des propriétés de "A" 94
Chapitre 6 100
Méthode générale d'implémentation d'un objet abstrait en termes d'un objet concret 100
6.1. Introduction 100
6.2. Implémentation d'une action en termes d'un objet concret 101
6.2.1. Réalisation d'une séquence 101
6.2.1.1. Construction de l'objet "ETAT" 101
6.2.1.2. Construction de la structure de conception 104
6.2.2. Représentation d'une boucle 106
6.2.2.1. Boucle dont la condition de fin porte sur les attributs de l'objet 107
6.2.2.2. Boucle qui nécessite un compteur 107
6.2.3. Représentation du "if ... then ... else ..." 111
6.2.3.1. Construction de l'objet "IF" 111
6.2.3.2. Construction de la structure de conception 111
6.3. Gestion de la concurrence entre les actions 114
6.3.1. Concurrence à l'intérieur d'un objet 114
6.3.2. Implémentation de deux actions concurrentes 116
6.3.2.1. Implémentation de deux actions qui entrent en conflit d'écriture au niveau de leurs attributs 116
6.3.2.2. Implémentation de deux actions qui entrent en conflit au niveau concret 116
6.4. Conclusion 120
Chapitre 7 122
Conclusion 122